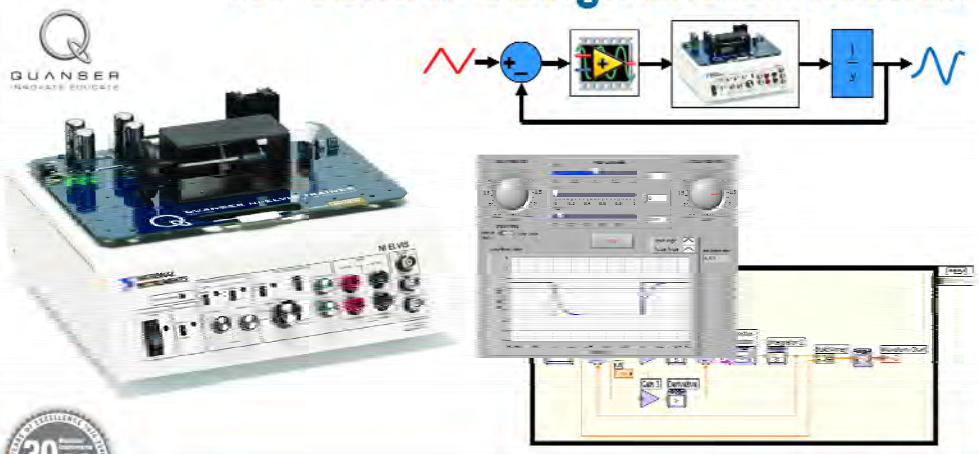
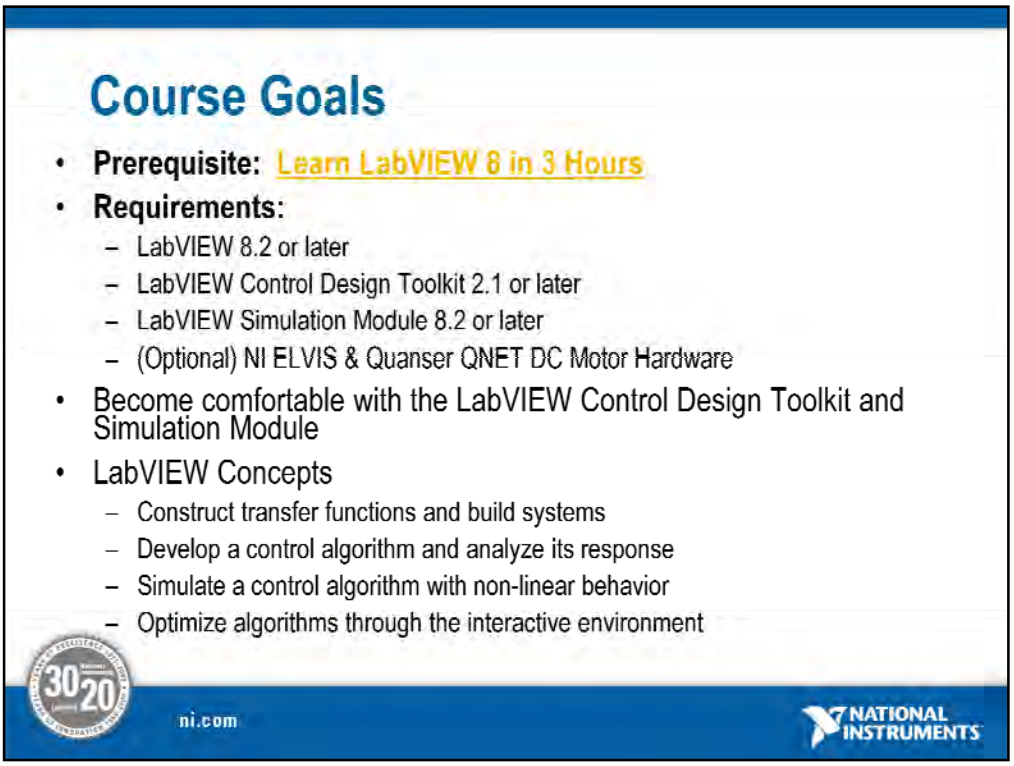
Introduction to LabVIEW in 3 Hours for Control Design and Simulation





Prerequisite: Learn LabVIEW 8 in 3 Hours - http://zone.ni.com/devzone/cda/tut/p/id/5247

This is a list of the objectives of the course.

This course prepares you to do the following:

- Become comfortable with the LabVIEW Control Design Toolkit and Simulation Module.
- Use LabVIEW to solve control problems.
- Construct transfer functions and build systems.
- Develop a control algorithm and analyze its response.
- Simulate a control algorithm with non-linear behavior.
- Optimize algorithms through the interactive environment.

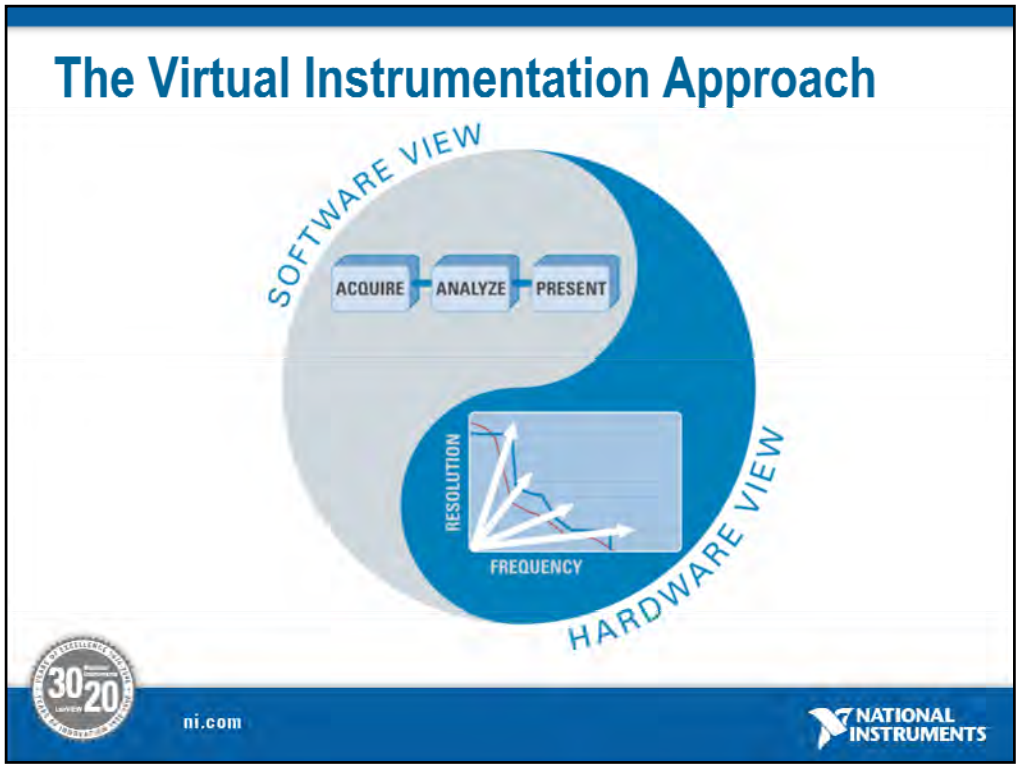
This course does *not* describe any of the following:

- Programming theory
- Every built-in LabVIEW function or object

NI does provide free reference materials on the above topics on ni.com.

The *LabVIEW Help* is also very helpful:

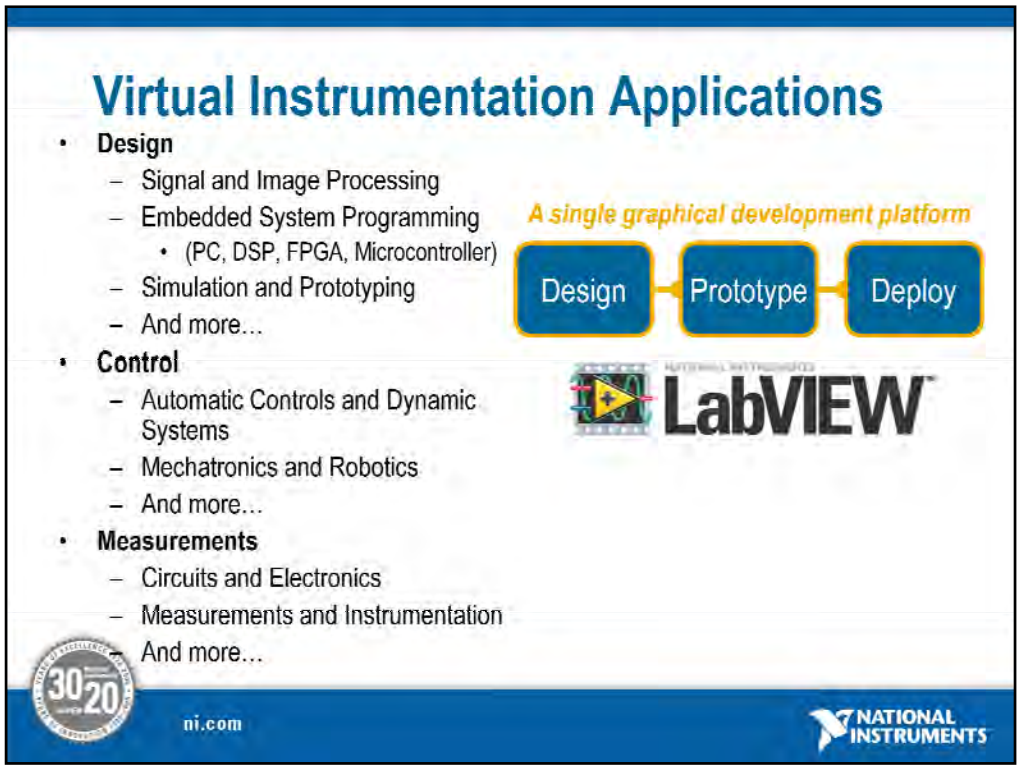
LabVIEW»Help»Search the LabVIEW Help...



Virtual Instrumentation

For more than 25 years, National Instruments has revolutionized the way engineers and scientists in industry, government, and academia approach measurement and automation. Leveraging PCs and commercial technologies, virtual instrumentation increases productivity and lowers costs for test, control, and design applications through easy-to-integrate software, such as NI LabVIEW, and modular measurement and control hardware for PXI, PCI, USB, and Ethernet.

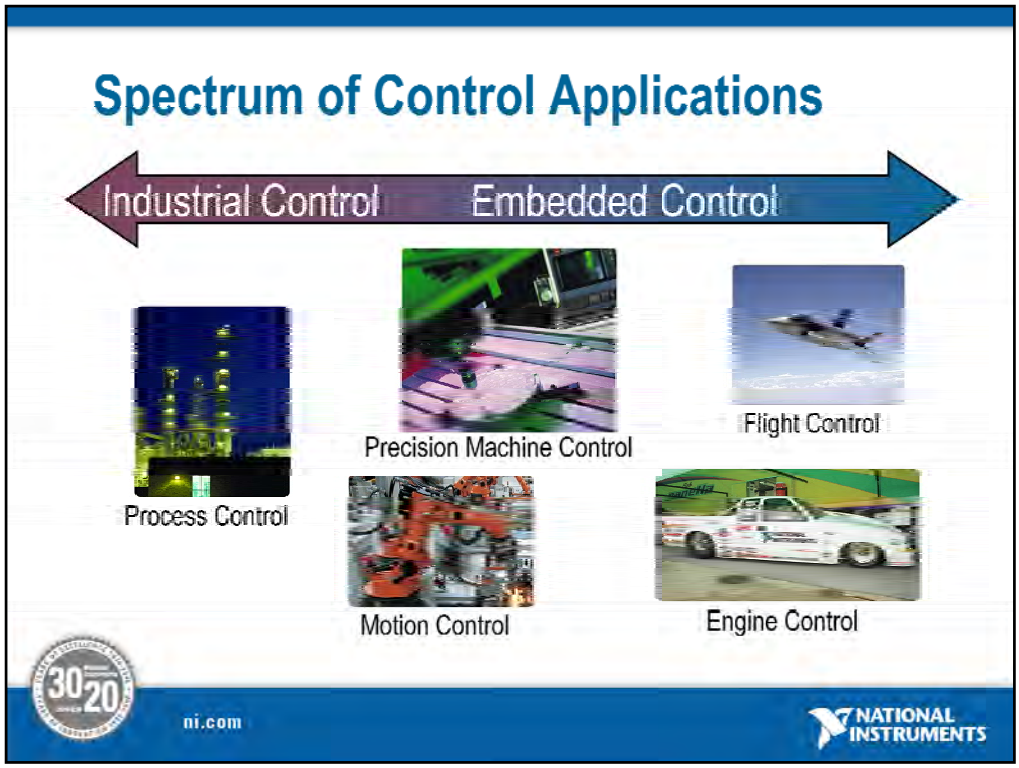
With virtual instrumentation, engineers use graphical programming software to create user-defined solutions that meet their specific needs, which is a great alternative to proprietary, fixed functionality traditional instruments. Additionally, virtual instrumentation capitalizes on the ever-increasing performance of personal computers. For example, in test, measurement, and control, engineers have used virtual instrumentation to downsize automated test equipment (ATE) while experiencing up to a 10 times increase in productivity gains at a fraction of the cost of traditional instrument solutions. Last year 25,000 companies in 90 countries invested in more than 6 million virtual instrumentation channels from National Instruments.



Virtual Instrumentation Applications

Virtual instrumentation is applicable in many different types of applications, starting from design to prototyping and deployment. The LabVIEW platform provides specific tools and models to solve specific applications ranging from designing signal processing algorithms to making voltage measurements and can target any number of platforms from the desktop to embedded devices – with an intuitive, powerful graphical paradigm.

With version 8, LabVIEW scales from design and development on PCs to several embedded targets from ruggedized toaster size prototypes to embedded systems on chips. LabVIEW streamlines system design with a single graphical development platform. In doing so, LabVIEW encompasses better management of distributed, networked systems because as the targets for LabVIEW grow varied and embedded, you will need to be able to more easily distribute and communicate between various LabVIEW code pieces in your system.



Control Applications

When discussing control systems, we are actually referring to a large range of applications. Probably even larger than you have suspected. First of all, control systems are commonly found in industrial environments. For example, consider an oil refinery with process control systems that continually manufacture and produce oil. The control system used for processing may consist of a Programmable Logic Controller (PLC) executing a PID algorithm, or a Distribute Control System (DCS) for a larger process control. In this case, the control system is used to manufacture a product.

A control system can also be part of an end product being manufactured. This has been seen primarily in the automotive and aerospace industries with electronic control units and flight control systems. However, control systems are now finding their way into other end products such as precision motor controllers for computer hard drives and white goods like washing machines.

While control systems used to manufacture a product often stem from established control strategies such as PID control, control systems embedded in end products often use new and innovative control strategies. The tools and techniques used to develop and embed control systems in end-products has evolved to include model-based design tools. However, manufacturing control engineers are also beginning to adopt these tools and techniques to develop more advanced control systems.

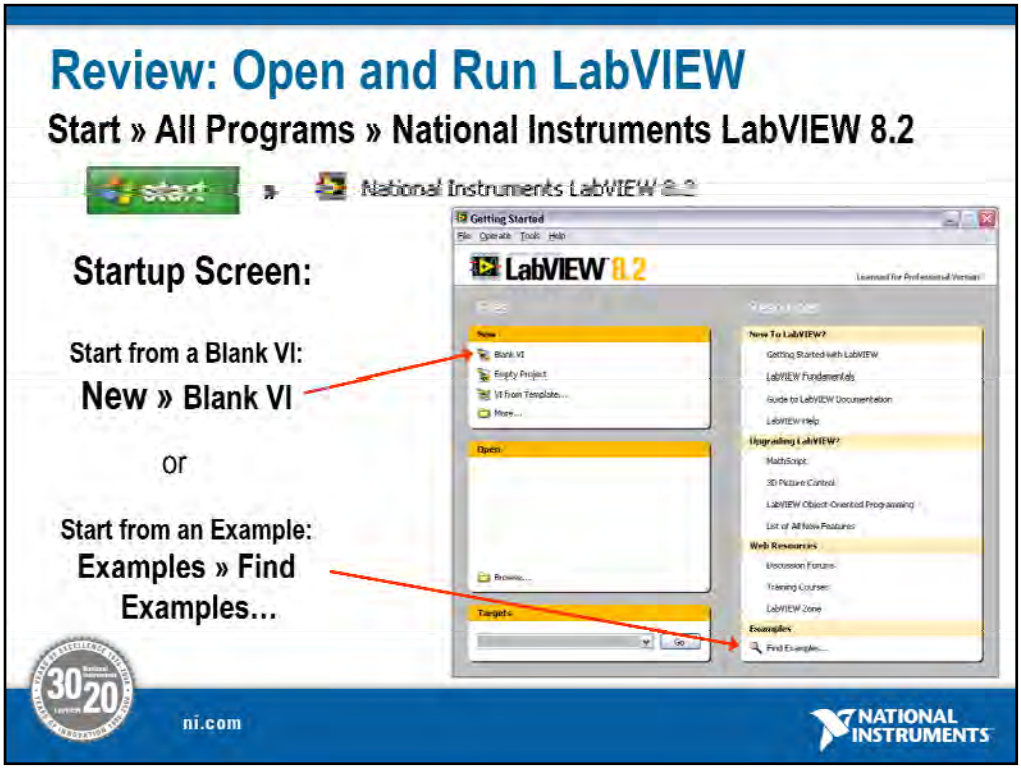
Topics Covered

A. Review: LabVIEW Environment

- Front Panel / Block Diagram
- Toolbar /Tools Palette

B. The Design Process

1. Modeling
2. Control Design
3. Simulation
4. Optimization
5. Deployment



LabVIEW

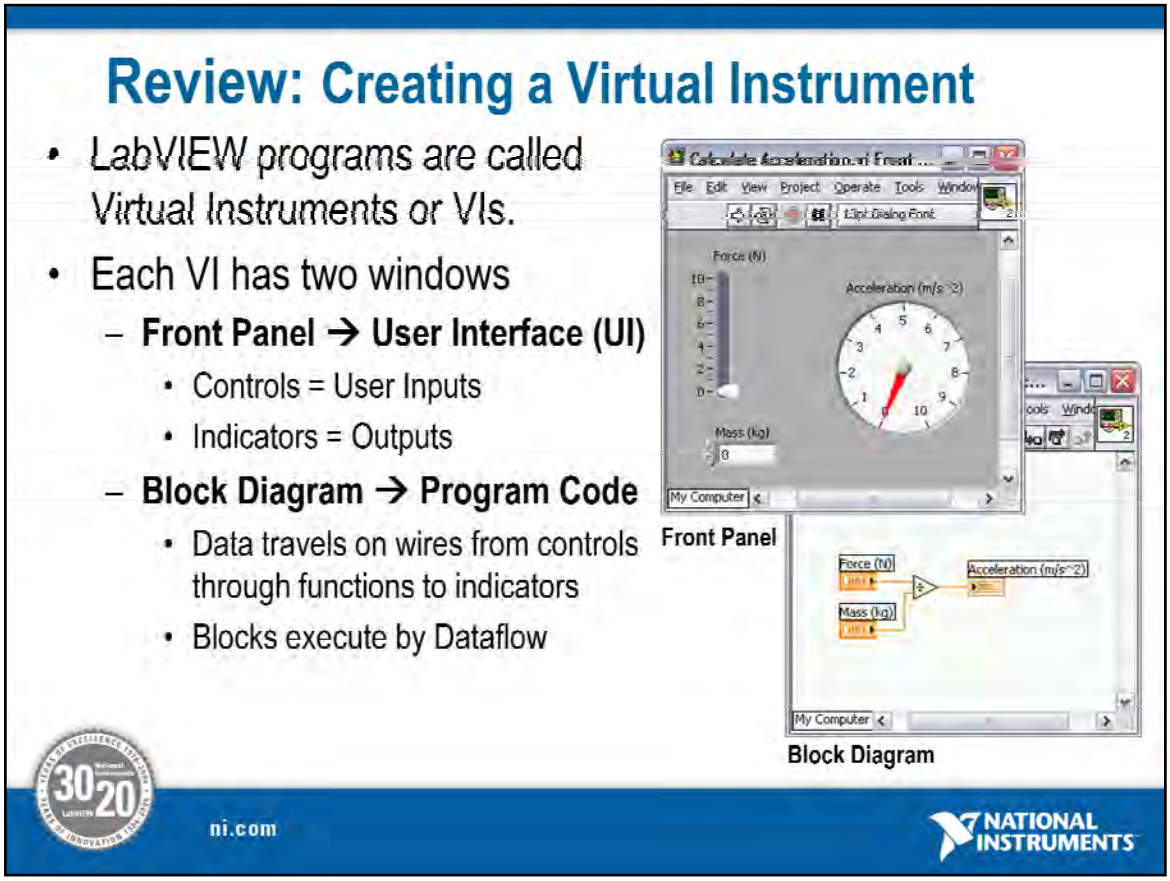
LabVIEW is a graphical programming language that uses icons instead of lines of text to create applications. In contrast to text-based programming languages, where instructions determine program execution, LabVIEW uses dataflow programming, where the flow of data determines execution order.

You can purchase several add-on software toolkits for developing specialized applications. All the toolkits integrate seamlessly in LabVIEW. Refer to the National Instruments Web site for more information about these toolkits.

LabVIEW also includes several wizards to help you quickly configure your DAQ devices and computer-based instruments and build applications.

LabVIEW Example Finder

LabVIEW includes hundreds of example VIs you can use and incorporate into VIs that you create. In addition to the example VIs that ship with LabVIEW, you also can access hundreds of example VIs on the NI Developer Zone (zone.ni.com). You can modify an example VI to fit an application, or you can copy and paste from one or more examples into a VI that you create.



LabVIEW programs are called virtual instruments (VIs).

Controls are inputs and indicators are outputs.

Each VI contains three main parts:

Front Panel – How the user interacts with the VI.

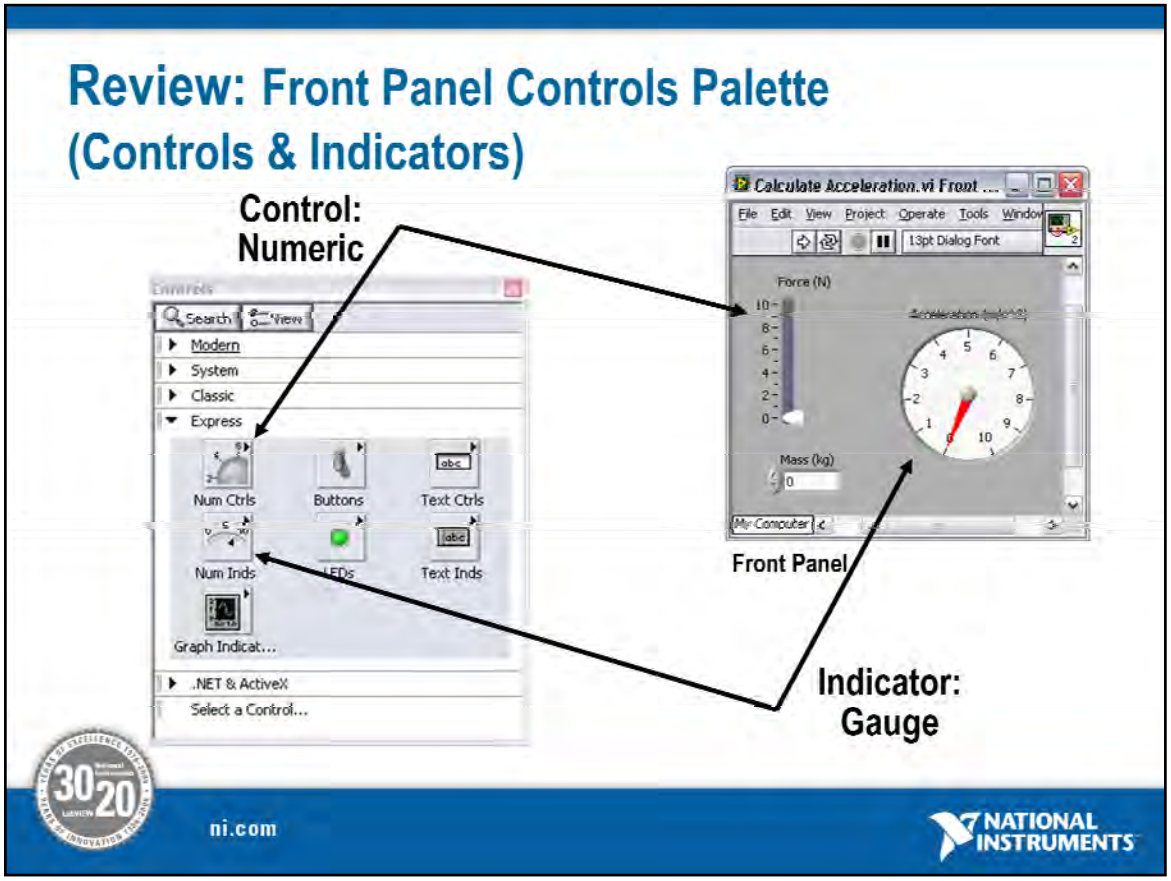
Block Diagram – The code that controls the program.

Icon/Connector – Means of connecting a VI to other VIs.

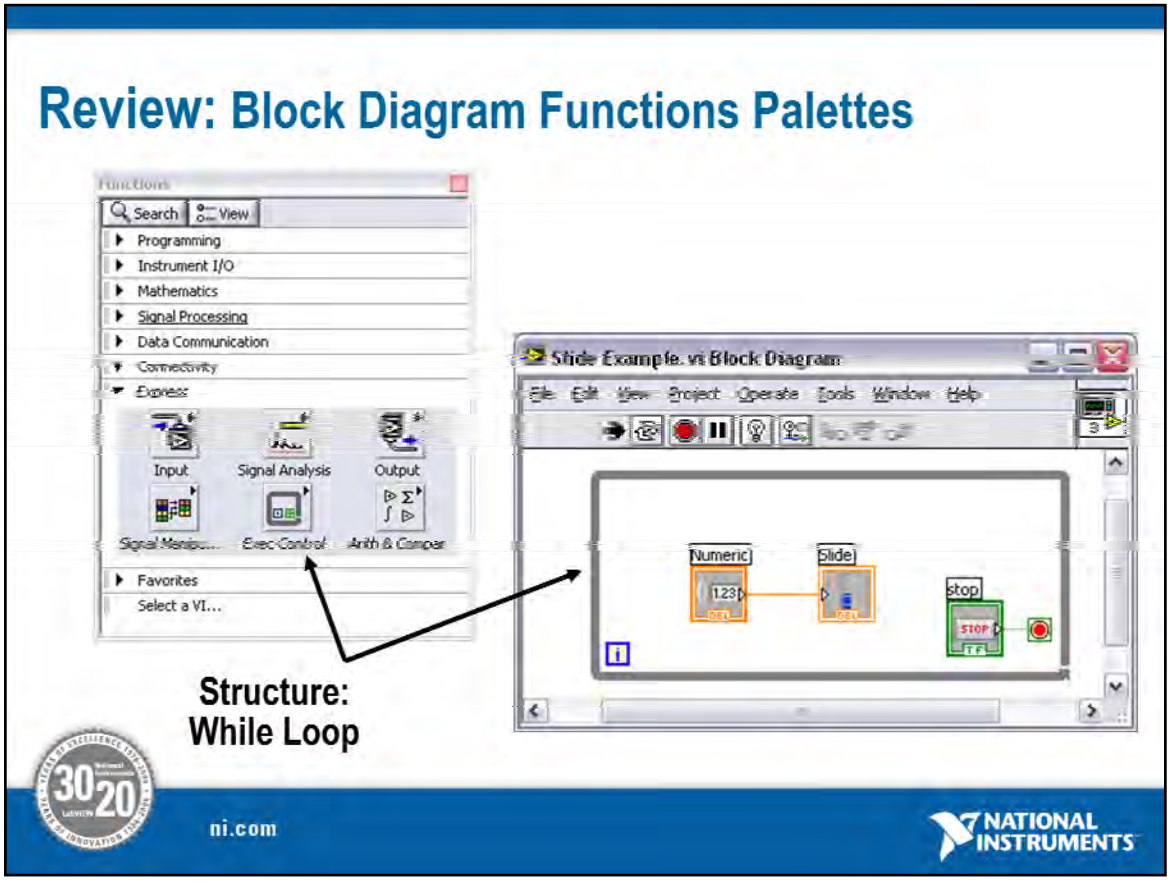
In LabVIEW, you build a user interface by using a set of tools and objects. The user interface is known as the front panel. You then add code using graphical representations of functions to control the front panel objects. The block diagram contains this code. In some ways, the block diagram resembles a flowchart.

Users interact with the Front Panel when the program is running. Users can control the program, change inputs, and see data updated in real time. Controls are used for inputs such as, adjusting a slide control to set an alarm value, turning a switch on or off, or to stop a program. Indicators are used as outputs. Thermometers, lights, and other indicators display output values from the program. These may include data, program states, and other information.

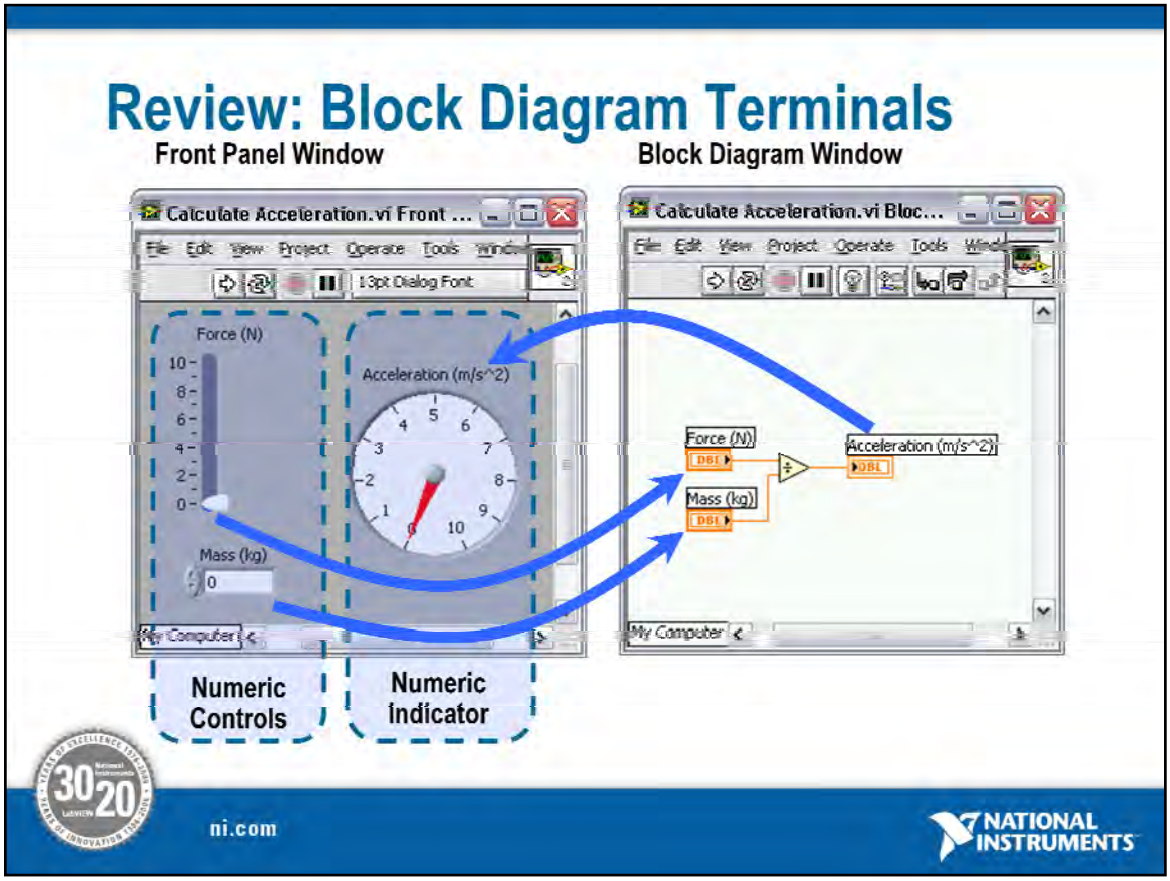
Every front panel control or indicator has a corresponding terminal on the block diagram. When a VI is run, values from controls flow through the block diagram, where they are used in the functions on the diagram, and the results are passed into other functions or indicators through wires.



Use the Controls palette to place controls and indicators on the front panel. The Controls palette is available only on the front panel. To view the palette, right-click on an open area in the front panel. Alternatively, select Window » Show Controls Palette. Tack down the Controls palette by clicking the pushpin on the top left corner of the palette.



Use the Functions palette to build the block diagram. The Functions palette is available only on the block diagram. To view the palette, select Window»Show Functions Palette. You also can display the Functions palette by right-clicking an open area on the block diagram. Tack down the Functions palette by clicking the pushpin on the top left corner of the palette.

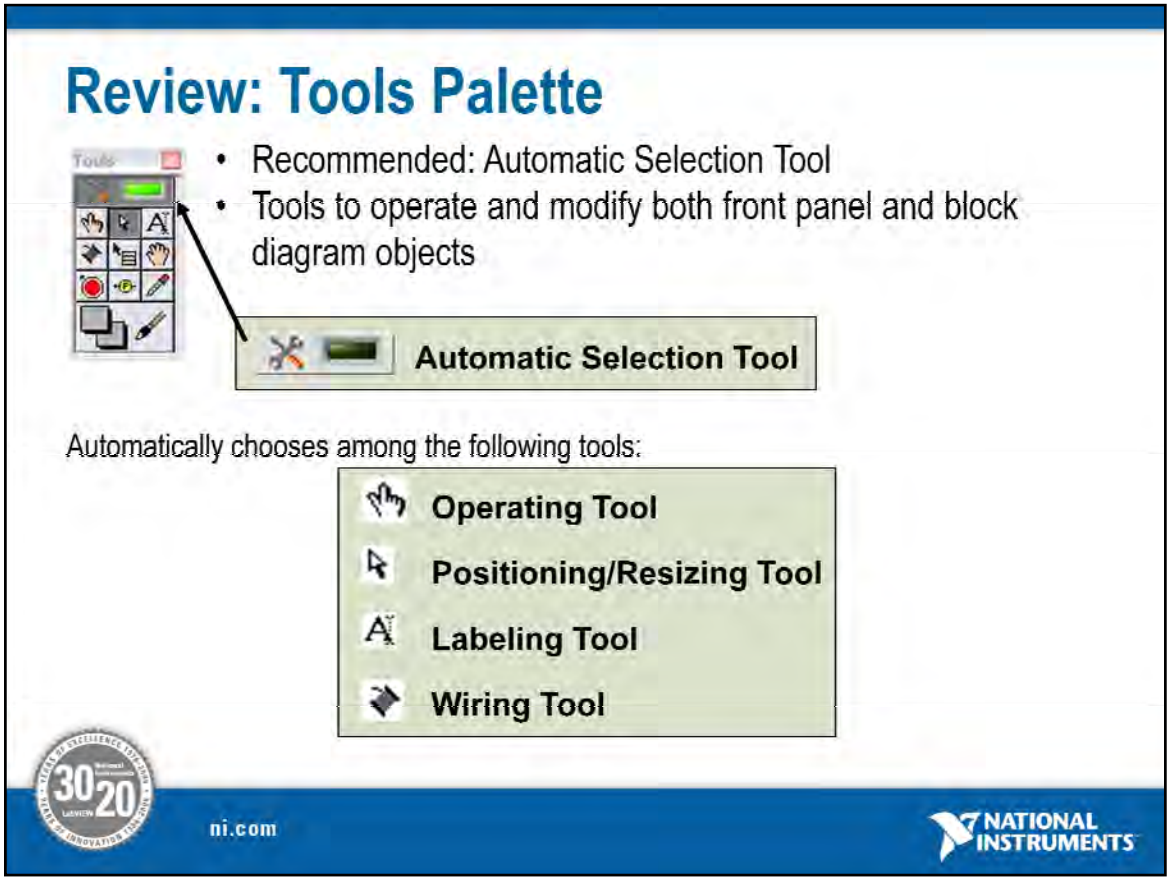


When you create an object on the Front Panel, a terminal will be created on the Block Diagram. These terminals give you access to the Front Panel objects from the Block Diagram code.

Each terminal contains useful information about the Front Panel object it corresponds to. For example, the color and symbols provide information about the data type. For example: The dynamic data type is a polymorphic data type represented by dark blue terminals. Boolean terminals are green with TF lettering.

In general, blue terminals should wire to blue terminals, green to green, and so on. This is not a hard-and-fast rule; LabVIEW will allow a user to connect a blue terminal (dynamic data) to an orange terminal (fractional value), for example. But in most cases, look for a match in colors.

Controls have an arrow on the right side and have a thick border. Indicators have an arrow on the left and a thin border. Logic rules apply to wiring in LabVIEW: Each wire must have one (but only one) source (or control), and each wire may have multiple destinations (or indicators).



If automatic tool selection is enabled and you move the cursor over objects on the front panel or block diagram, LabVIEW automatically selects the corresponding tool from the Tools palette. Toggle automatic tool selection by clicking the Automatic Tool Selection button in the Tools palette.

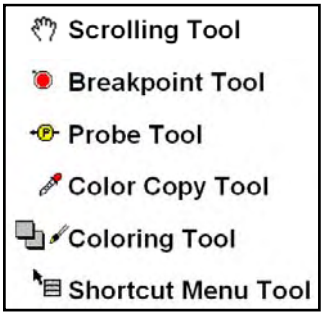
Use the Operating tool to change the values of a control or select the text within a control.

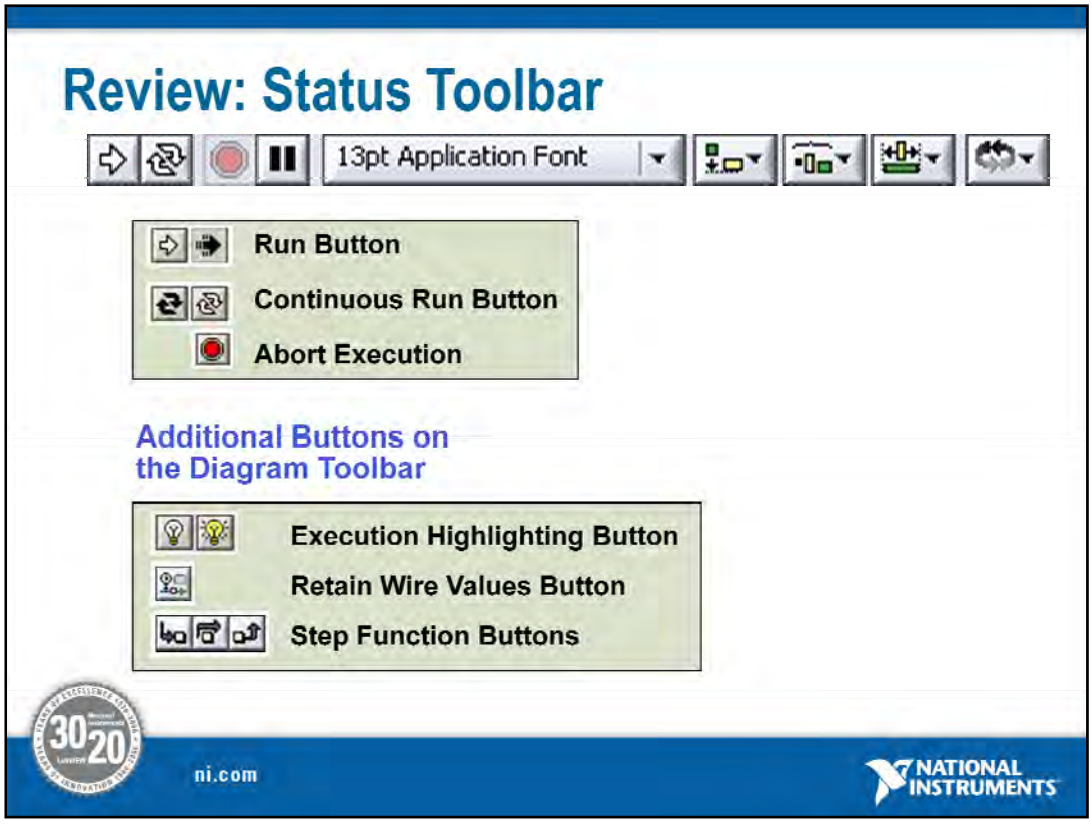
Use the Positioning tool to select, move, or resize objects. The Positioning tool changes shape when it moves over a corner of a resizable object.

Use the Labeling tool to edit text and create free labels. The Labeling tool changes to a cursor when you create free labels.

Use the Wiring tool to wire objects together on the block diagram.

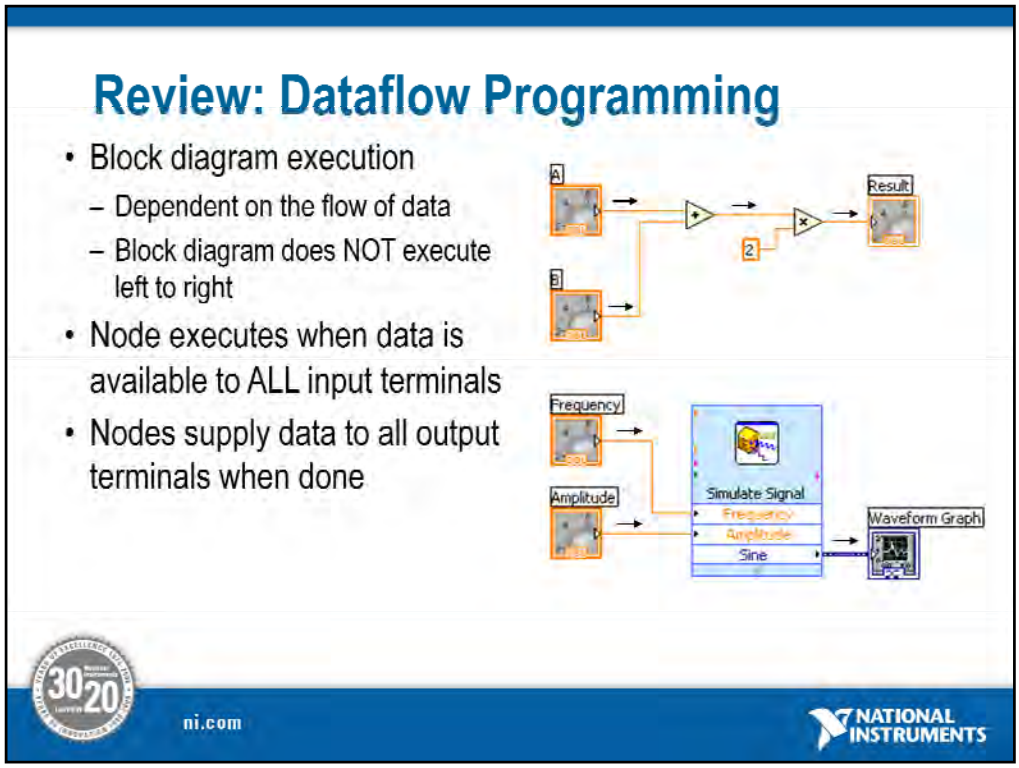
Other important tools:





- Click the Run button to run the VI. While the VI runs, the Run button appears with a black arrow if the VI is a top-level VI, meaning it has no callers and therefore is not a subVI.
- Click the Continuous Run button to run the VI until you abort or pause it. You also can click the button again to disable continuous running.
- While the VI runs, the Abort Execution button appears. Click this button to stop the VI immediately.

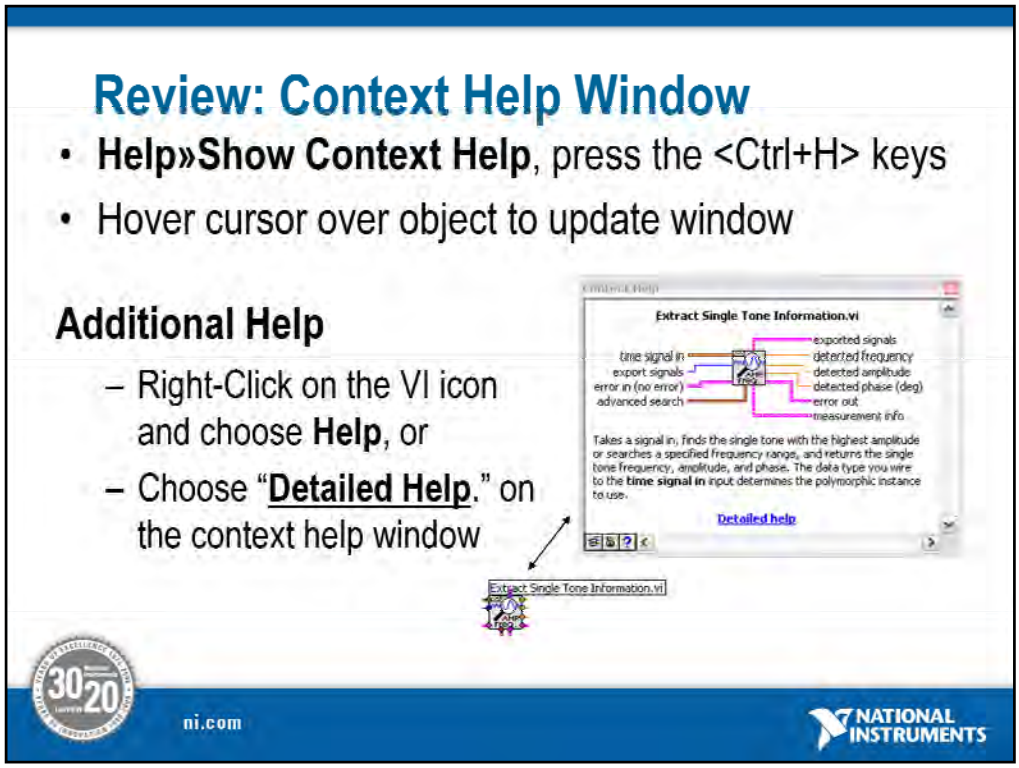
  Note: Avoid using the Abort Execution button to stop a VI. Either let the VI complete its data flow or design a method to stop the VI programmatically. By doing so, the VI is at a known state. For example, place a button on the front panel that stops the VI when you click it.
- Click the Pause button to pause a running VI. When you click the Pause button, LabVIEW highlights on the block diagram the location where you paused execution. Click the Pause button again to continue running the VI.
- Select the Text Settings pull-down menu to change the font settings for the VI, including size, style, and color.
- Select the Align Objects pull-down menu to align objects along axes, including vertical, top edge, left, and so on.
- Select the Distribute Objects pull-down menu to space objects evenly, including gaps, compression, and so on.
- Select the Resize Objects pull-down menu to change the width and height of front panel objects.



LabVIEW follows a dataflow model for running VIs. A block diagram node executes when all its inputs are available. When a node completes execution, it supplies data to its output terminals and passes the output data to the next node in the dataflow path. Visual Basic, C++, JAVA, and most other text-based programming languages follow a control flow model of program execution. In control flow, the sequential order of program elements determines the execution order of a program.

Consider the block diagram above. It adds two numbers and then multiplies by 2 from the result of the addition. In this case, the block diagram executes from left to right, not because the objects are placed in that order, but because one of the inputs of the Multiply function is not valid until the Add function has finished executing and passed the data to the Multiply function. Remember that a node executes only when data are available at all of its input terminals, and it supplies data to its output terminals only when it finishes execution. In the second piece of code, the Simulate Signal Express VI receives input from the controls and passes its result to the Graph.

You may consider the add-multiply and the simulate signal code to co-exist on the same block diagram in parallel. This means that they will both begin executing at the same time and run independent of one another. If the computer running this code had multiple processors, these two pieces of code could run independent of one another (each on its own processor) without any additional coding.



The Context Help window displays basic information about LabVIEW objects when you move the cursor over each object. Objects with context help information include VIs, functions, constants, structures, palettes, properties, methods, events, and dialog box components.

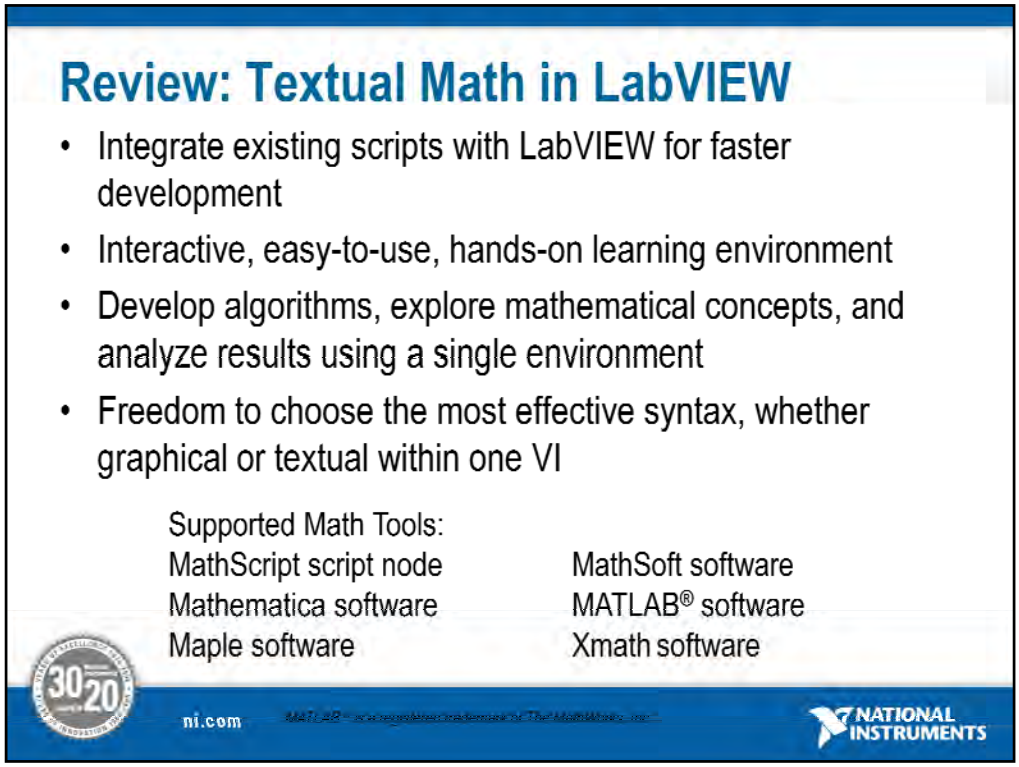
To display the Context Help window, select Help»Show Context Help, press the <Ctrl+H> keys, or press the Show Context Help Window button in the toolbar

Connections displayed in Context Help:

Required – bold
Recommended – normal
Optional – dimmed

Additional Help

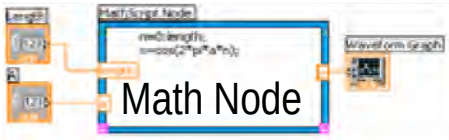
- VI, Function, & How-To Help is also available.
  - Help» VI, Function, & How-To Help
  - Right-click the VI icon and choose Help, or
  - Choose "Detailed Help." on the context help window.
- LabVIEW Help – reference style help
  - Help»Search the LabVIEW Help…



Overview

With the release of National Instruments LabVIEW 8, you have new freedom to choose the most effective syntax for technical computing, whether you are developing algorithms, exploring DSP concepts, or analyzing results. You can instrument your scripts and develop algorithms on the block diagram by interacting with popular third-party math tools such as The MathWorks Inc. MATLAB software, Mathematica, Maple, Mathcad, IDL and Xmath. Use of these math tools with LabVIEW is achieved in a variety of ways depending on the vendor as listed below:

Native LabVIEW textual math node:
MathScript node, Formula node

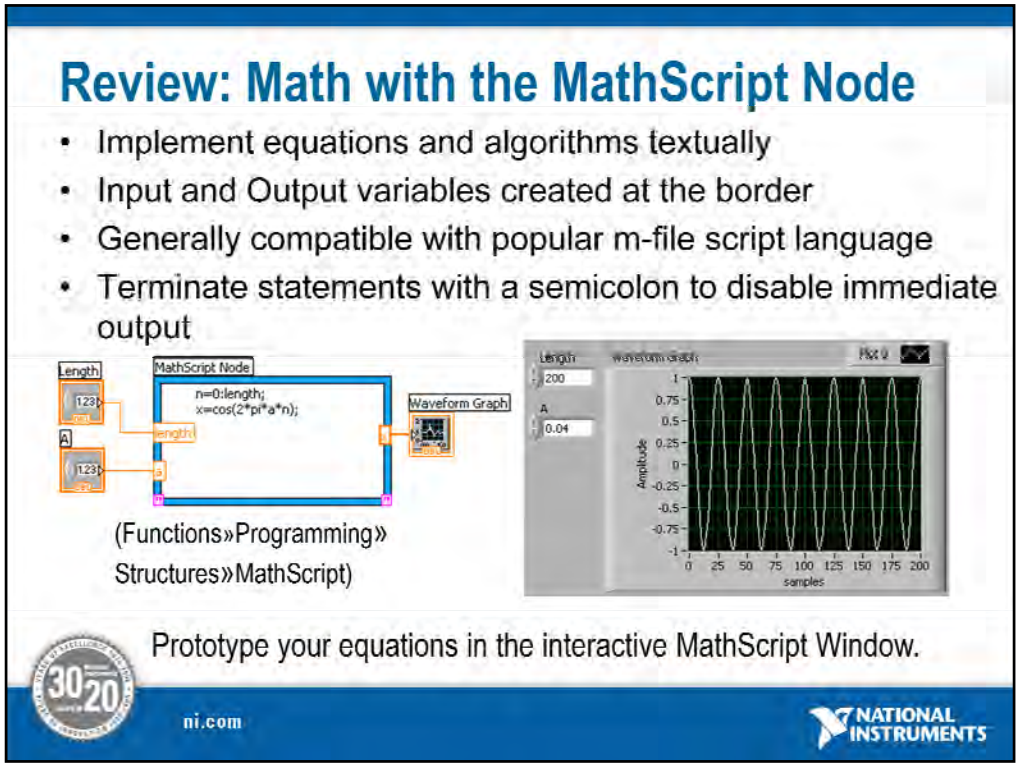


Communication with vendor software through LabVIEW node:
Xmath node, MATLAB script node, Maple* node, IDL* node

Communication with vendor software through VI Server:
Mathematica* VIs, and Mathcad* VIs

In LabVIEW 8, you can combine the intuitive LabVIEW graphical dataflow programming with MathScript, a math-oriented textual programming language that is generally compatible with popular m-file script language.

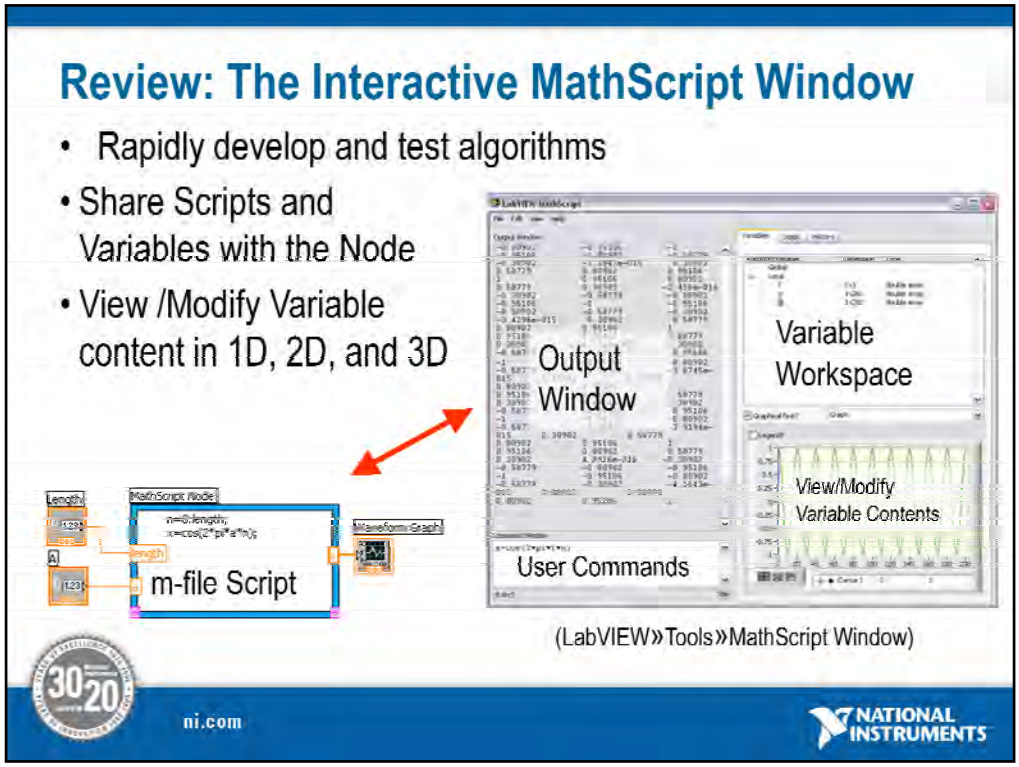
*LabVIEW toolkit specific to the math tool must be installed.



The MathScript Node enhances LabVIEW by adding a native text-based language for mathematical algorithm implementation in the graphical programming environment. M-file scripts you've written and saved from the MathScript window can be opened and used in the MathScript node. M-file scripts you created in other math software will generally run as well. The MathScript allows you to pick the syntax you are most comfortable with to solve the problem. Equations can be instrumented with the MathScript Node for parameter exploration, simulation, or deployment in a final application.

The MathScript Node:

- Located in the Programming»Structures subpalette.
- Resizable box for entering textual computations directly into block diagrams.
- To add variables, right-click and choose Add Input or Add Output.
- Name variables as they are used in formula. (Names are case sensitive.)
- The data type of the output can be changed by right-clicking the input or output node.
- Statements should be terminated with a semicolon to suppress output.
- Ability to import & export m-files by right-clicking on the node.



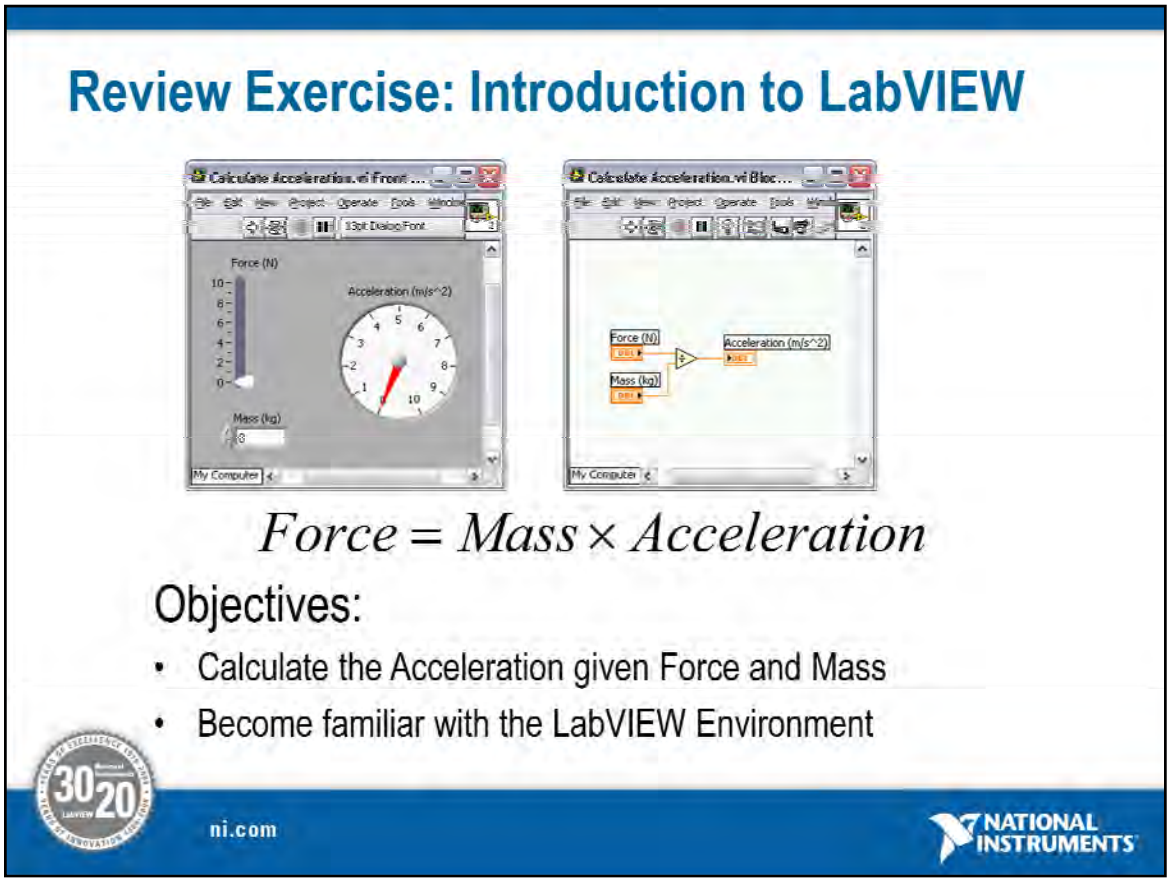
The MathScript Window provides an interactive environment where equations can be prototyped and calculations can be made. The MathScript Window and Node share a common syntax and global variables making the move from prototype to implementation seamless. The data preview pane provides a convenient way to view variable data as numbers, graphically, or audibly (with soundcard support).

Help for MathScript

Help for the environment can be accessed using the Mathscript Interactive Environment Window. Type **Help** in the command window for an introduction to MathScript help. **Help** followed by a **function** will display help specific to that function.

Features of the interactive MathScript Window:

- Prototype equations and formulas through the command Window
- Easily access function help by typing **Help <function>** in the Command Window
- Select a variable to display its data in the Preview Pane and even listen to the result
- Write, Save, Load, and Run m-files using the Script tab
- Share data between the MathScript Node in LabVIEW and the MathScript Window using Global Variables
- Advanced plotting features and image export features



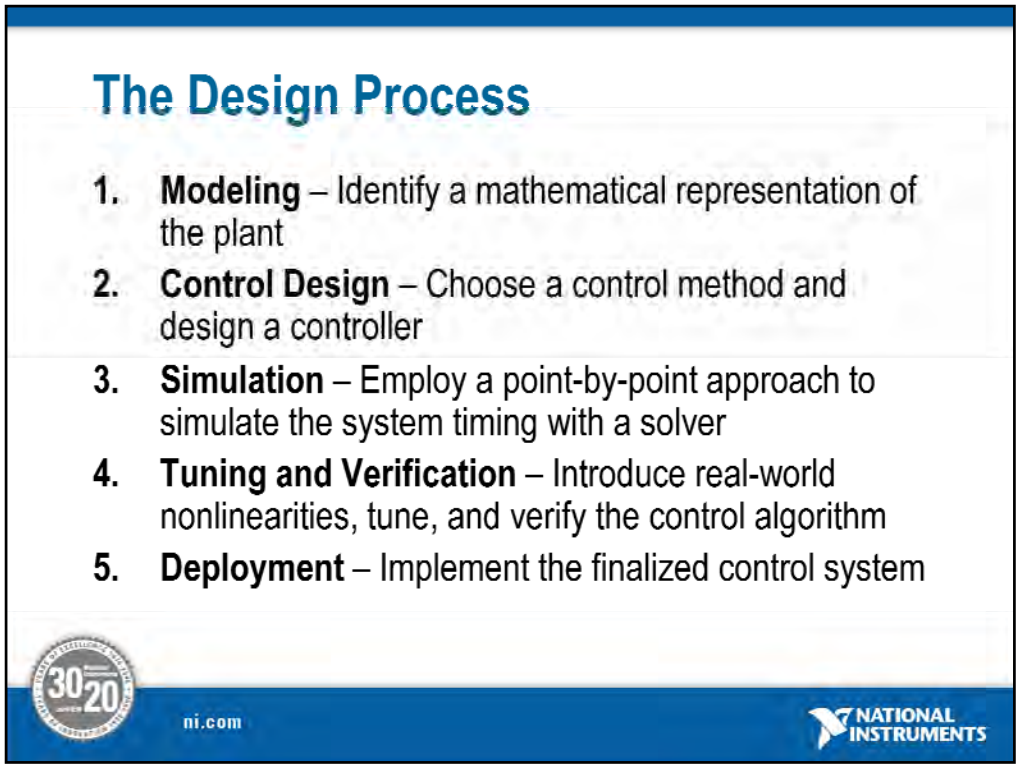
Review Exercise: Introduction to LabVIEW

To begin our model, first we will make a simple model calculation of Newton's classic F=m*a equation. Since we are interested in using Force and Mas as inputs, we will calculate a=F/m.

1. Create two front panel controls: a numeric control and a vertical slider.
    a) Right click on the front panel to bring up the Controls Palette
    b) Navigate to the Modern » Numeric sub-palette by left-clicking
    c) Left click the Numeric control.
    d) Left click on the Front Panel to place the control.
    e) The name of the control will be highlighted. Start typing to rename this control to "Mass (kg)"
    f) Repeat steps a-d with a Vertical Pointer Slide. Name this control "Force (N)"
2. Create a front panel gauge indicator.
    a) In the Controls Palette, navigate to the **Modern » Numeric** sub-palette.
    b) Select the Gauge indicator
    c) Place and name this indicator "Acceleration (m/s^2)"
3. Notice how corresponding terminals appeared on the white block diagram.
4. Place a divide function on the block diagram
    a) In the Functions Palette, navigate to the **Programming » Numeric** sub-palette
    b) Select and place the Divide function.

5. Wire the controls and indicators to the Divide function
    a) Hover the mouse on the right side of the Force control. The cursor will change to the wiring tool automatically.
    b) Left click to begin a wire
    c) Move the mouse to the top input of the divide function
    d) Left-click to complete the wire
    e) Repeat steps 1-4 to wire the Mass control to the Divide function's bottom input and the output to the Acceleration indicator.

End of Exercise



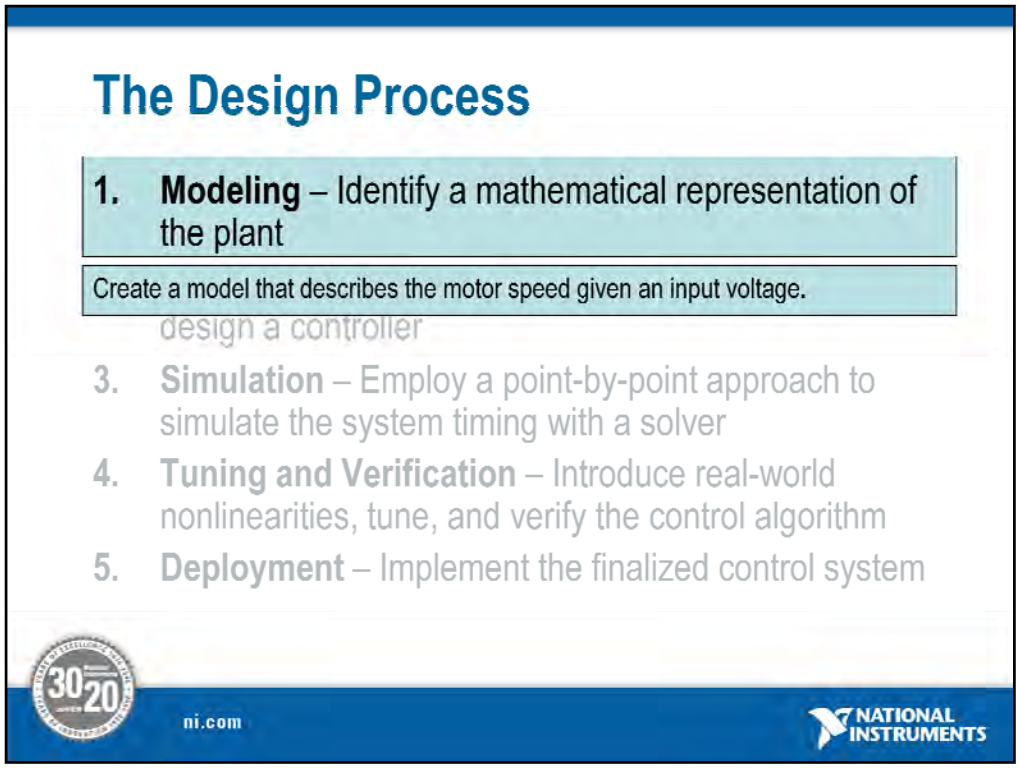
The controller can be optimized by fully understanding the plant. This understanding comes from analysis done in the LabVIEW Control Design Toolkit. Specialized graphs, such as Bode, root-locus, and Nyquist plot, builds intuition of how the plant will behave. Further analysis using these plots with an open loop system with the controller allow stability and other characteristics to be calculated. In this way, we use Control Design to optimize the parameters, in our case the gains P, I, and D. Graphs in the time domain, such as the step response, provide immediate feedback on the ideal behavior of the system, such as rise time, overshoot, settling time, and steady-state error.

We then use the Simulation Module to validate these parameters under real world constraints. One prime example is the maximum and minimum voltage and current available to be supplied to the motor. The voltage limits of the motor define the linear range of operation. Outside of that range, the motor might be damaged or become unpredictable. This is an example of nonlinear behavior, which cannot be modeled using LTI analysis found in the Control Design Toolkit. The Simulation Module allows nonlinearities to be represented in the model. Voltage limits can be modeled using the Saturation block. Other nonlinear behaviors can be written in LabVIEW, MathScript, and the LabVIEW Formula Node, and integrated into the model as a Sub VI.

The optimization process usually includes several iterations between Control Design and Simulation to achieve the desired performance of the system.

Step 1:
Modeling

Identify a mathematical representation of the plant



The controller can be optimized by fully understanding the plant. This understanding comes from analysis done in the LabVIEW Control Design Toolkit. Specialized graphs, such as Bode, root-locus, and Nyquist plot, builds intuition of how the plant will behave. Further analysis using these plots with an open loop system with the controller allow stability and other characteristics to be calculated. In this way, we use Control Design to optimize the parameters, in our case the gains P, I, and D. Graphs in the time domain, such as the step response, provide immediate feedback on the ideal behavior of the system, such as rise time, overshoot, settling time, and steady-state error.

We then use the Simulation Module to validate these parameters under real world constraints. One prime example is the maximum and minimum voltage and current available to be supplied to the motor. The voltage limits of the motor define the linear range of operation. Outside of that range, the motor might be damaged or become unpredictable. This is an example of nonlinear behavior, which cannot be modeled using LTI analysis found in the Control Design Toolkit. The Simulation Module allows nonlinearities to be represented in the model. Voltage limits can be modeled using the Saturation block. Other nonlinear behaviors can be written in LabVIEW, MathScript, and the LabVIEW Formula Node, and integrated into the model as a Sub VI.

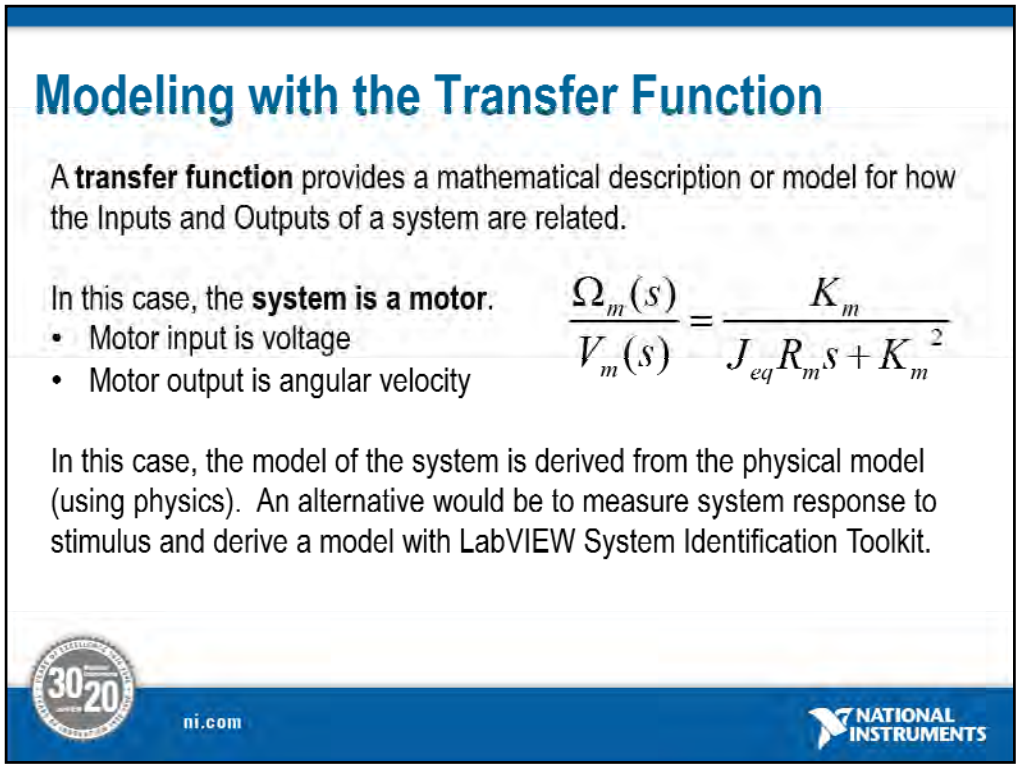
The optimization process usually includes several iterations between Control Design and Simulation to achieve the desired performance of the system.

Why Model?

A model represents a physical system
- Inputs and outputs have real-world units
- It is often a simplification of the system's behavior

Many types of systems can be modeled
- Mechanical systems
- Electronic circuits
- Analog and digital filters
- Thermal and fluid systems

Models can be represented in many ways
- Transfer function
- State space
- Others



Transfer function is the mathematical relationship between the output of the control system i.e. the plant and its input. Generally represented by the Laplace transform of the output of the system or plant divided by the Laplace transform of the plant's input under the zero initial condition. We can also represent transfer function by Z-Transform.

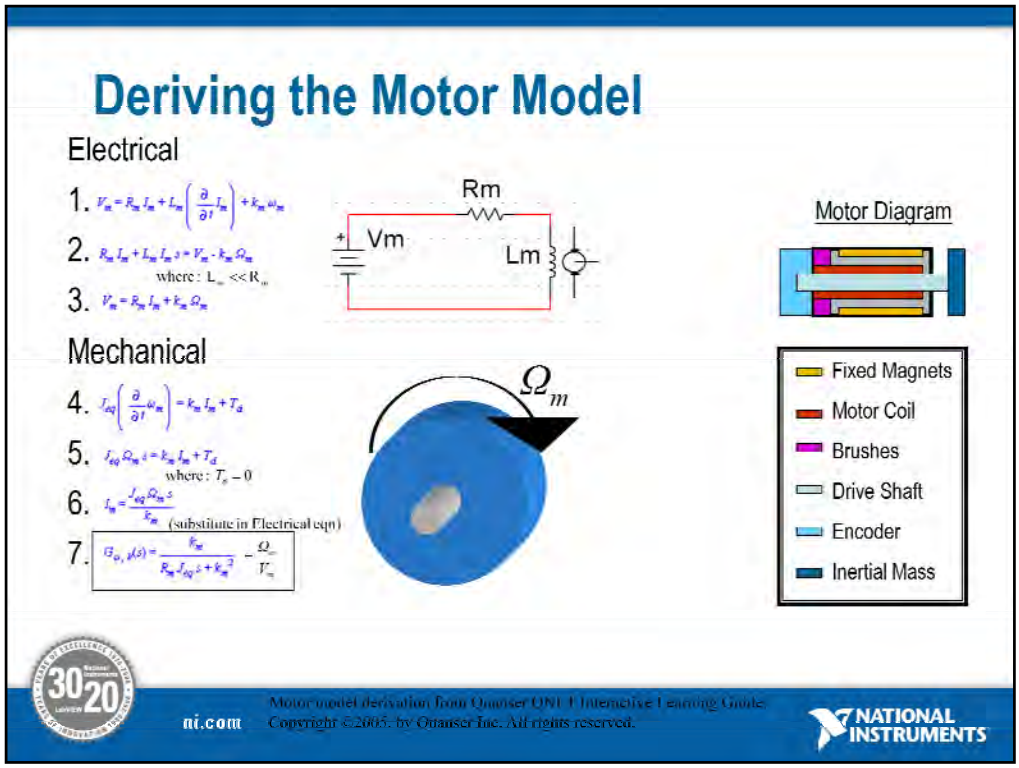
The input and the output of a transfer function can be either mechanical or electrical parameters, in this case our system is motor, and the input to the system is voltage and the output from the system is angular velocity.

We can derive a transfer function of the physical model by using physics equations i.e. equations of motion, and so on; or can be estimated using tools like LabVIEW System Identification Toolkit.

QNET DC Motor Control Trainer

The model in this course is based on the **Quanser QNET DC Motor**.

- The motor is a 24 volt DC motor
- The motor is highly linear in terms of its speed given a voltage
- We will use the Control Design Toolkit functions to create the mathematical model in LabVIEW



The model for the DC motor is derived from the electrical and mechanical characteristics of the system. The input to the motor is voltage (V_m), and the output from the motor is angular velocity (ω_m). The transfer function that defines the motor model is the ratio between the Laplace transforms of the input and output:

$(\Omega_m/V_m)=k_m/(R_m*J_{eq}*s+k_m^2)$

Where:

k_m = Motor back-EMF constant (V/(rad/s))

R_m = Motor armature resistance (Ohms)

J_{eq} = Equivalent moment of inertia (kg*m^2) (Assume that $J_{eq}=J_m$(Motor armature moment of inertia))

This transfer function makes up the Plant model that will be used throughout the exercises in this course. The Plant model will be used to design a Controller model, which can then be tested with the actual motor.

Motor Specifications Sheet

Symbol	Description	Value	Unit
Motor:			
R_m	Motor armature resistance	3.30	ohms
K_t	Motor torque constant	0.0280	N.m
K_m	Motor back-emf constant	0.0280	V/(rad/s)
J_m	Moment of inertia of motor rotor	9.64e-6	$kg.m^2$
M_l	Inertial load disc mass	0.033	kg
r_l	Inertial load disc radius	0.0242	m
Pulse-Width Modulated Amplifier:			
V_{max}	PWM amplifier maximum output voltage	24	V
	PWM amplifier maximum output current	5	A
	PWM amplifier gain	2.3	V/V

$$Plant(Motor): \quad V_m \rightarrow \frac{K_m}{J_{eq}R_m s + K_m^{\,2}} \rightarrow \Omega_m$$

V_m – Input Voltage

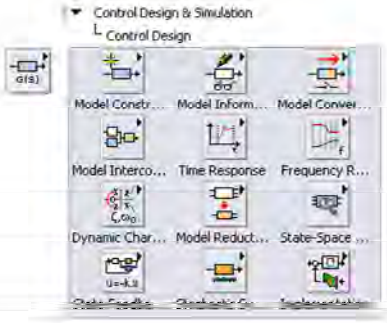
Ω_m – Angular Velocity

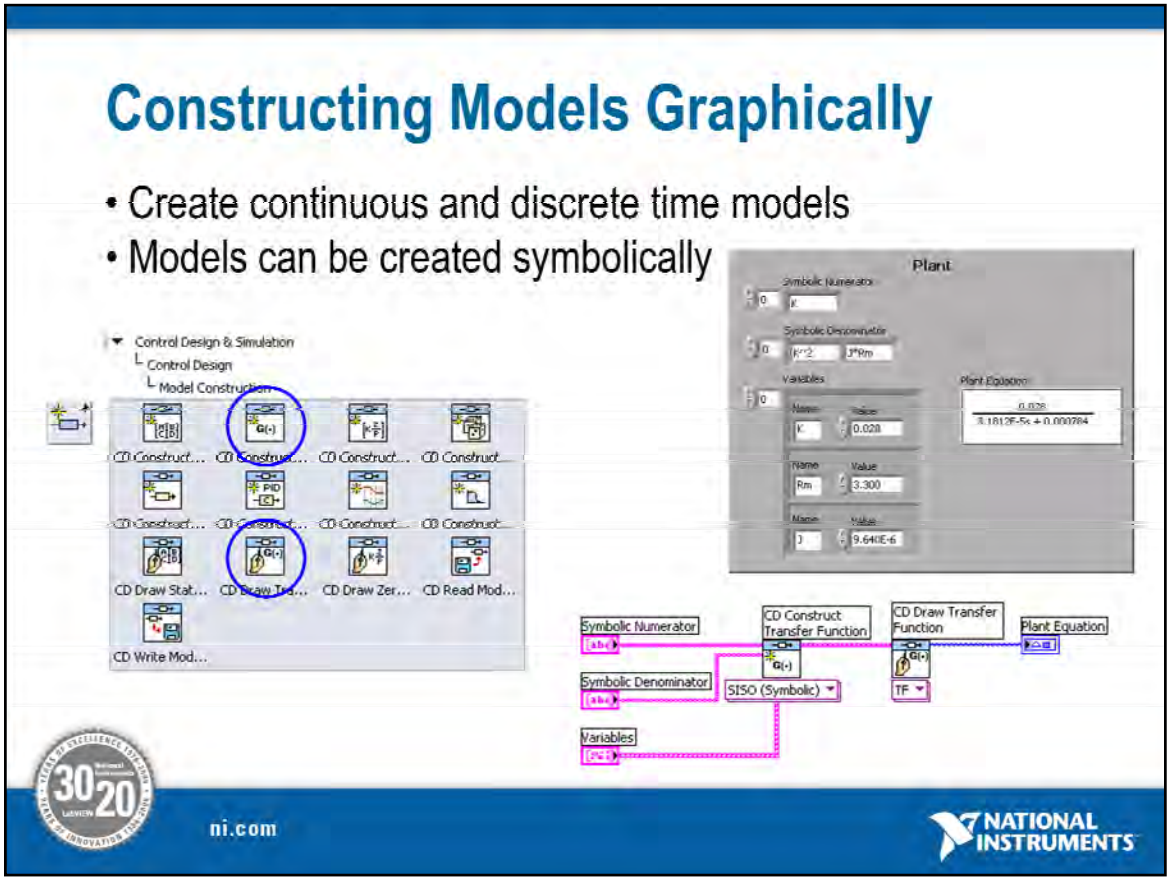
Motor model specifications from Quanser QNET Interactive Learning Guide.
Copyright © 2005, by Quanser Inc. All rights reserved.

NATIONAL INSTRUMENTS

LabVIEW Control Design Toolkit

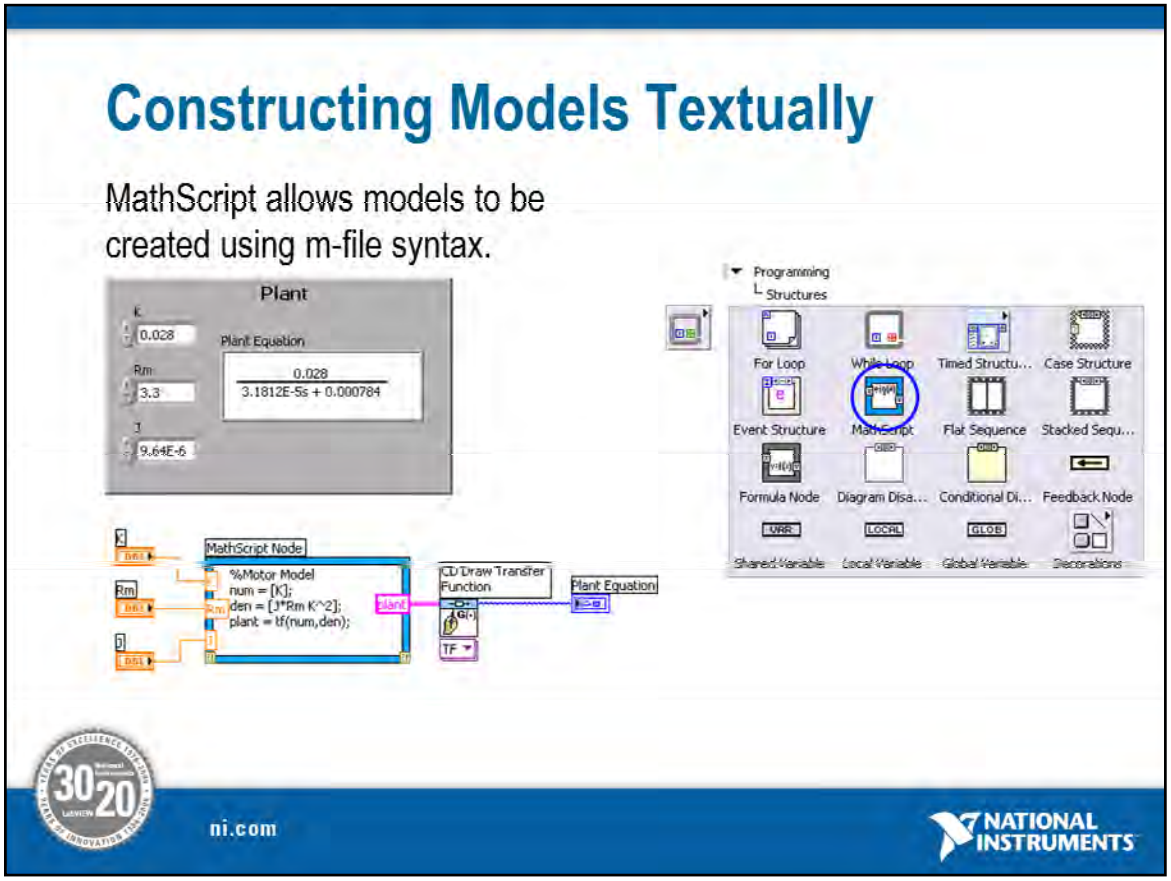
- Construct and analyze system models
- Design basic and advanced control algorithms
- Simulate response of controller designs
- Analyze control efficiency and stability interactively



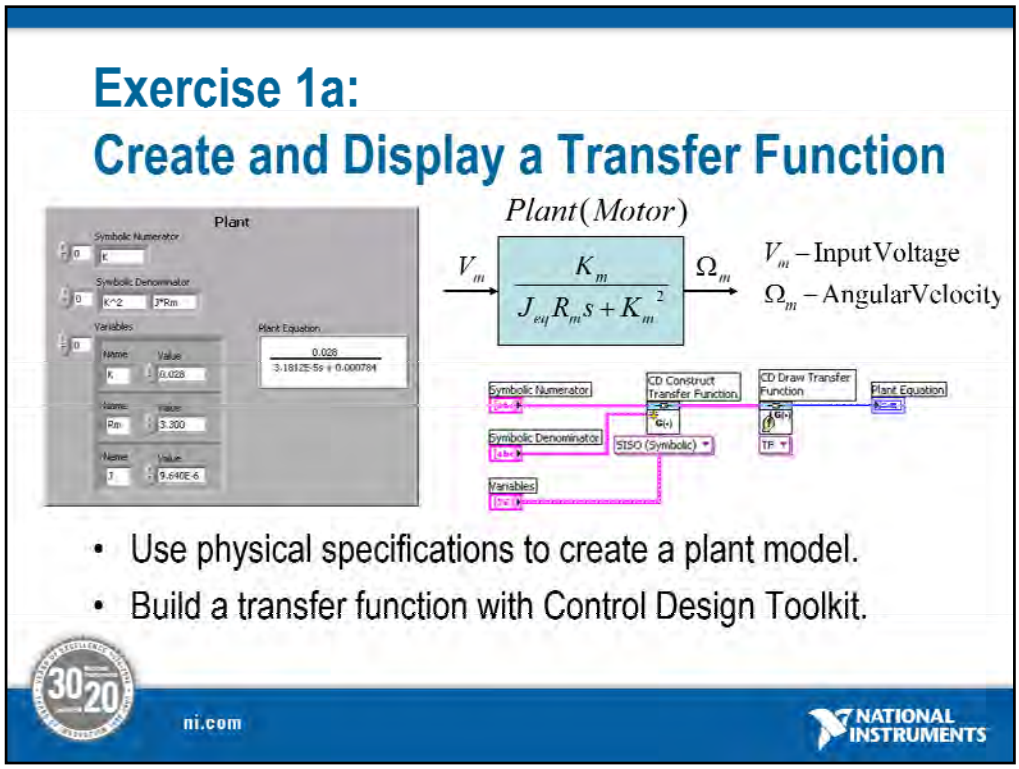


Use the Model Construction VIs to graphically create linear system models and modify the properties of a system model. You also can use the Model Construction VIs to save a system model to a file, read a system model from a file, or obtain a visual representation of a model.

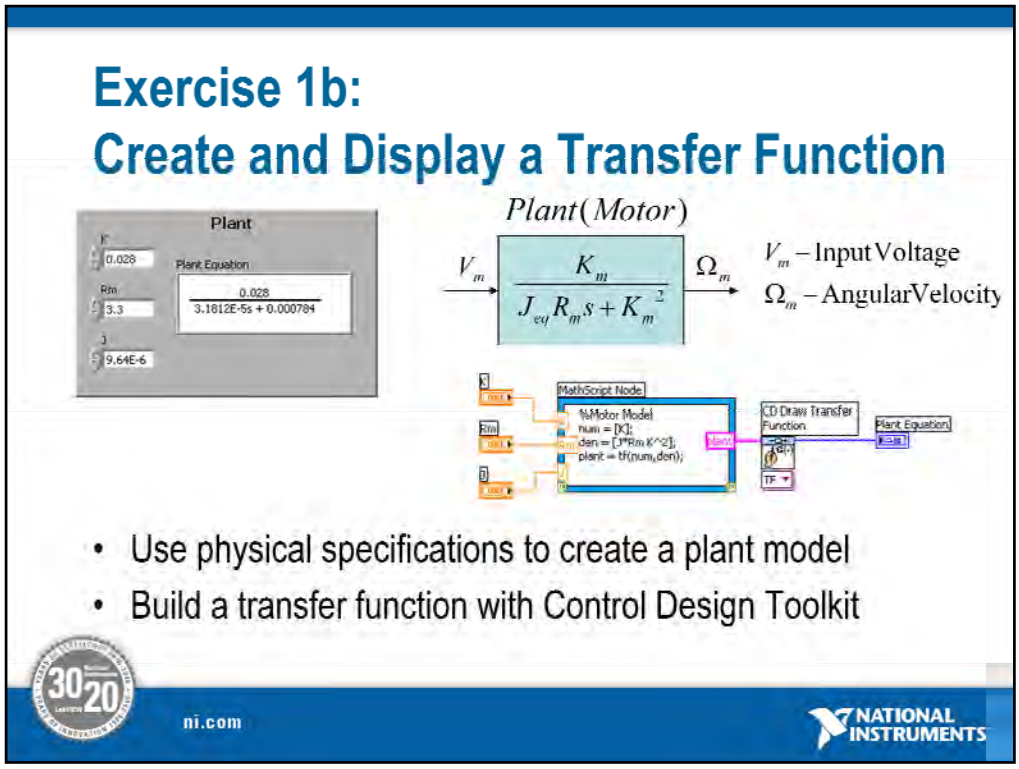
The first step in working with the Control Design toolkit is inputting a model to work with. The Control Design Toolkit can work with Transfer Function (TF), State Space (SS), and Zero Pole Gain (ZPK) models. The model itself is stored as a cluster which can be passed from VI to VI to perform multiple operations on a single model.



Models can be constructed using a text-based language with the Control Design MathScript functions. Use the MathScript Node to define a region on the block diagram for textual programming. MathScript can work with Transfer Function (TF), State Space (SS), and Zero Pole Gain (ZPK) models. The cluster containing the model information produced by the MathScript Node has the same form as the cluster produced by the Model Construction VIs, allowing for interchangeable use of textual and graphical programming.



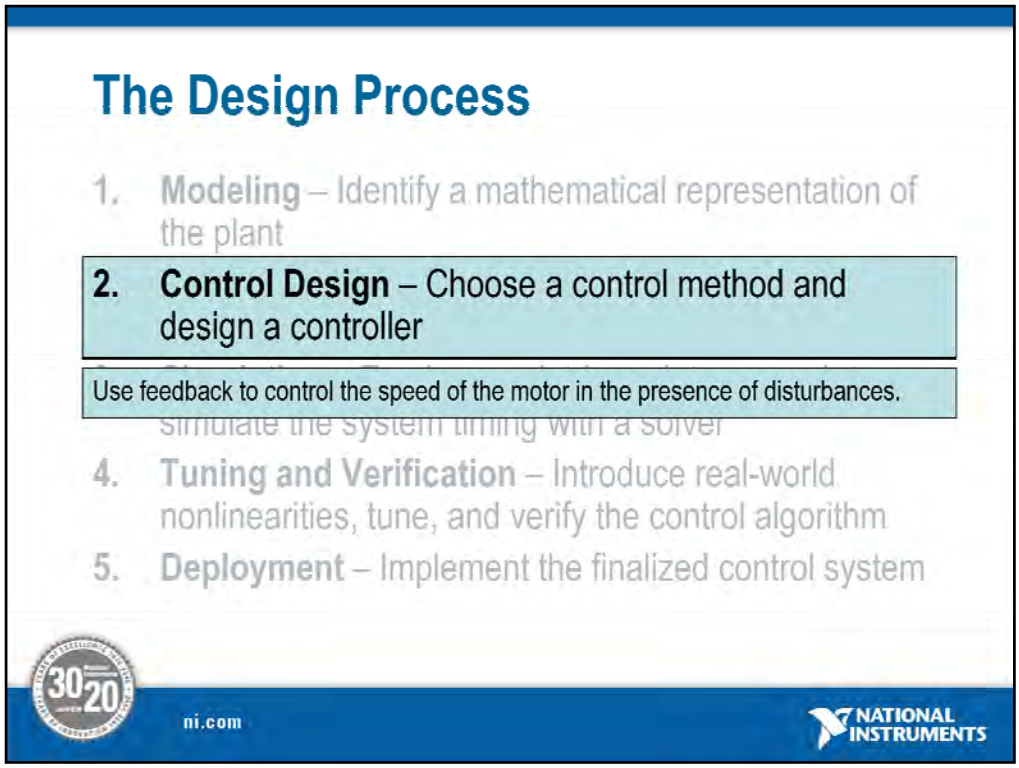
Exercise Instructions can be found at the end of PDF version of this manual or independently as “Updated Exercise Instructions.pdf”.



Exercise Instructions can be found at the end of PDF version of this manual or independently as “Updated Exercise Instructions.pdf”.

Step 2:
Control Design

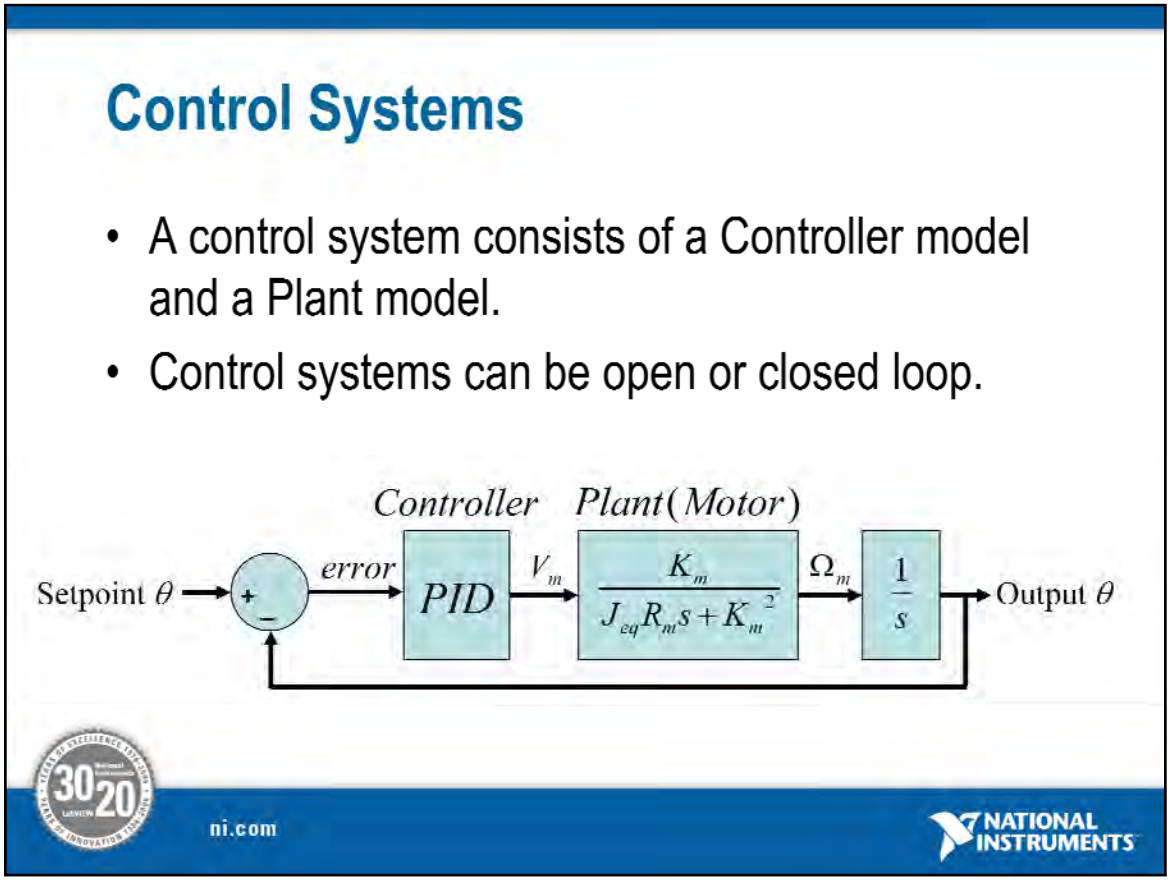
Choose a control type and design a controller



The controller can be optimized by fully understanding the plant. This understanding comes from analysis done in the LabVIEW Control Design Toolkit. Specialized graphs, such as Bode, root-locus, and Nyquist plot, builds intuition of how the plant will behave. Further analysis using these plots with an open loop system with the controller allow stability and other characteristics to be calculated. In this way, we use Control Design to optimize the parameters, in our case the gains P, I, and D. Graphs in the time domain, such as the step response, provide immediate feedback on the ideal behavior of the system, such as rise time, overshoot, settling time, and steady-state error.

We then use the Simulation Module to validate these parameters under real world constraints. One prime example is the maximum and minimum voltage and current available to be supplied to the motor. The voltage limits of the motor define the linear range of operation. Outside of that range, the motor might be damaged or become unpredictable. This is an example of nonlinear behavior, which cannot be modeled using LTI analysis found in the Control Design Toolkit. The Simulation Module allows nonlinearities to be represented in the model. Voltage limits can be modeled using the Saturation block. Other nonlinear behaviors can be written in LabVIEW, MathScript, and the LabVIEW Formula Node, and integrated into the model as a Sub VI.

The optimization process usually includes several iterations between Control Design and Simulation to achieve the desired performance of the system.

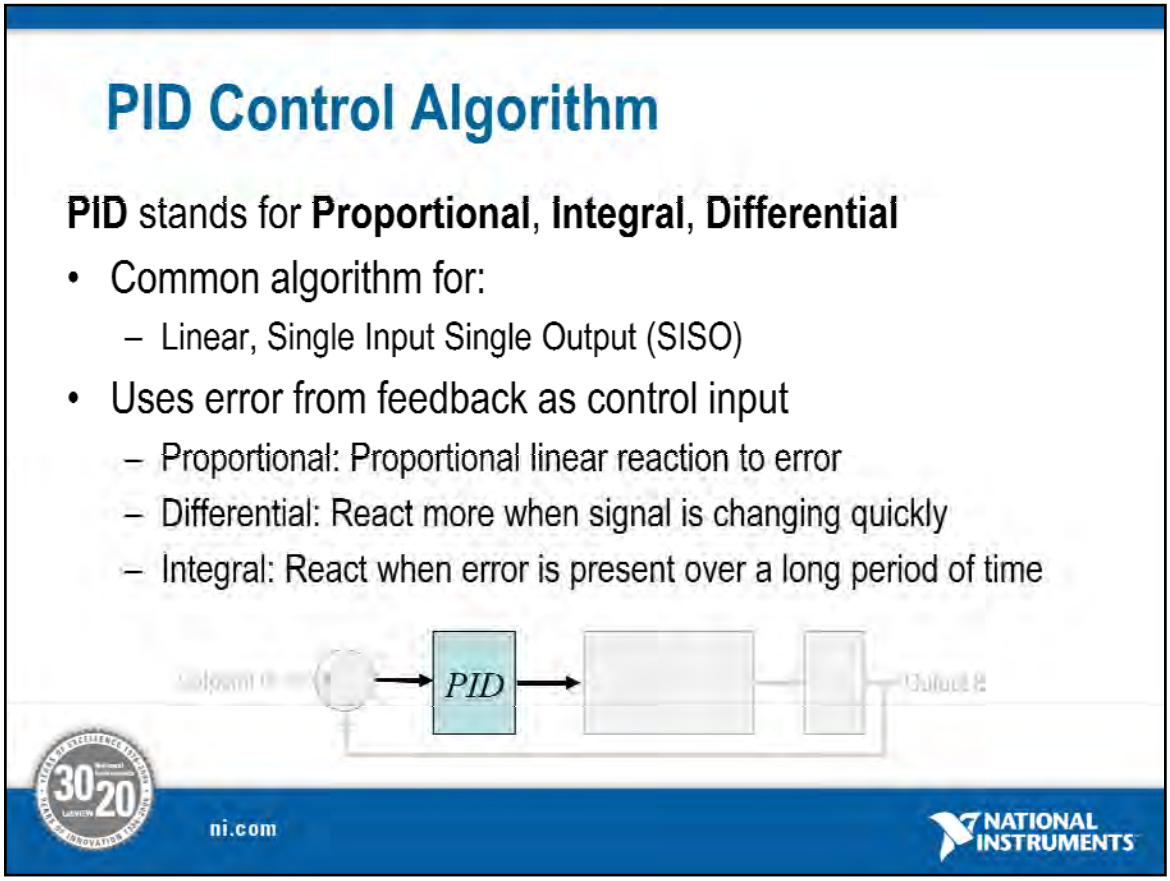


A common control system consists of a controller model and a plant model. The output of the controller is sent to the plant. In a closed loop system, the output of the plant is subtracted from the input (set point) of the system, producing an error value, which acts as the controller input. This is known as feedback.

1. Plant (Motor) Model
    - The plant model is a mathematical representation of the system in question. In this case, the plant is a motor.
    - The input to the motor is voltage (V_m), and the output from the motor is angular velocity in radians per second (ω_m).

2. Controller Model: PID
    - The controller model contains a mathematical algorithm that supplies an input to the plant model based on the error.
    - PID (Proportional, Integral, Derivative) is a common algorithm used in control systems.
    - The input to the PID controller is error (setpoint – output) in radians (θ). The output from the PID controller is voltage (V_m).

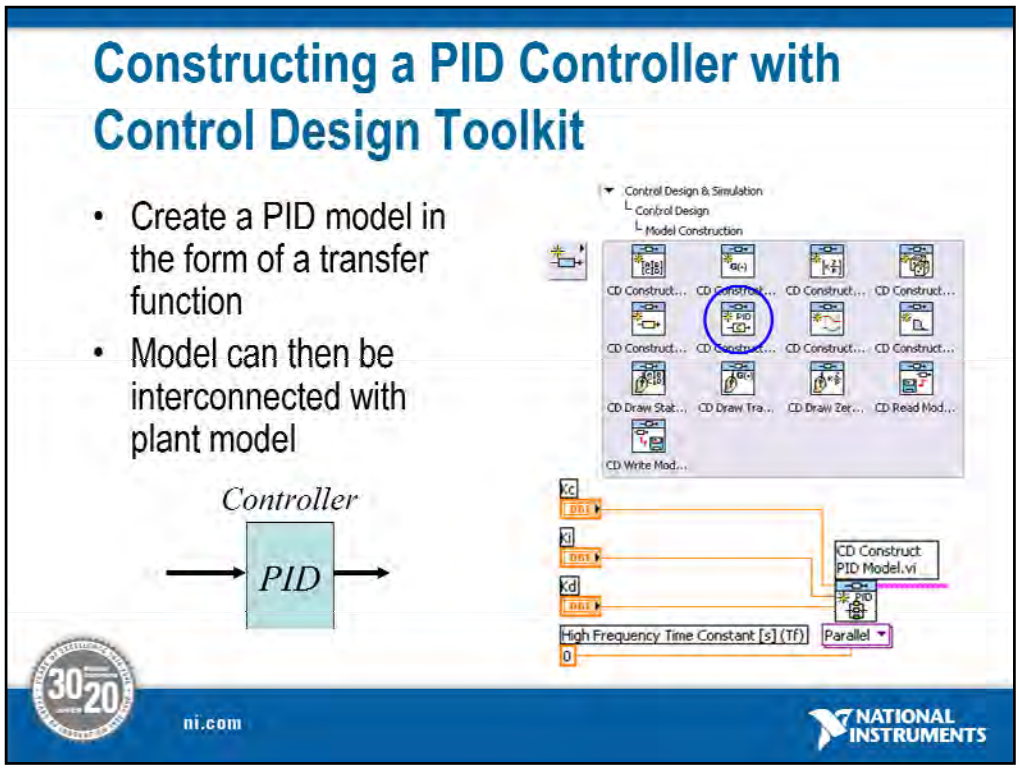
3. Integrator: 1/s
    - The integrator is used to convert the output from the motor plant (angular velocity) to have units consistent with the setpoint of the system (angular position).



PID controller provides the excitation for the plant; the gains K_p, K_d and K_i PID controller is set to control the overall system behavior.

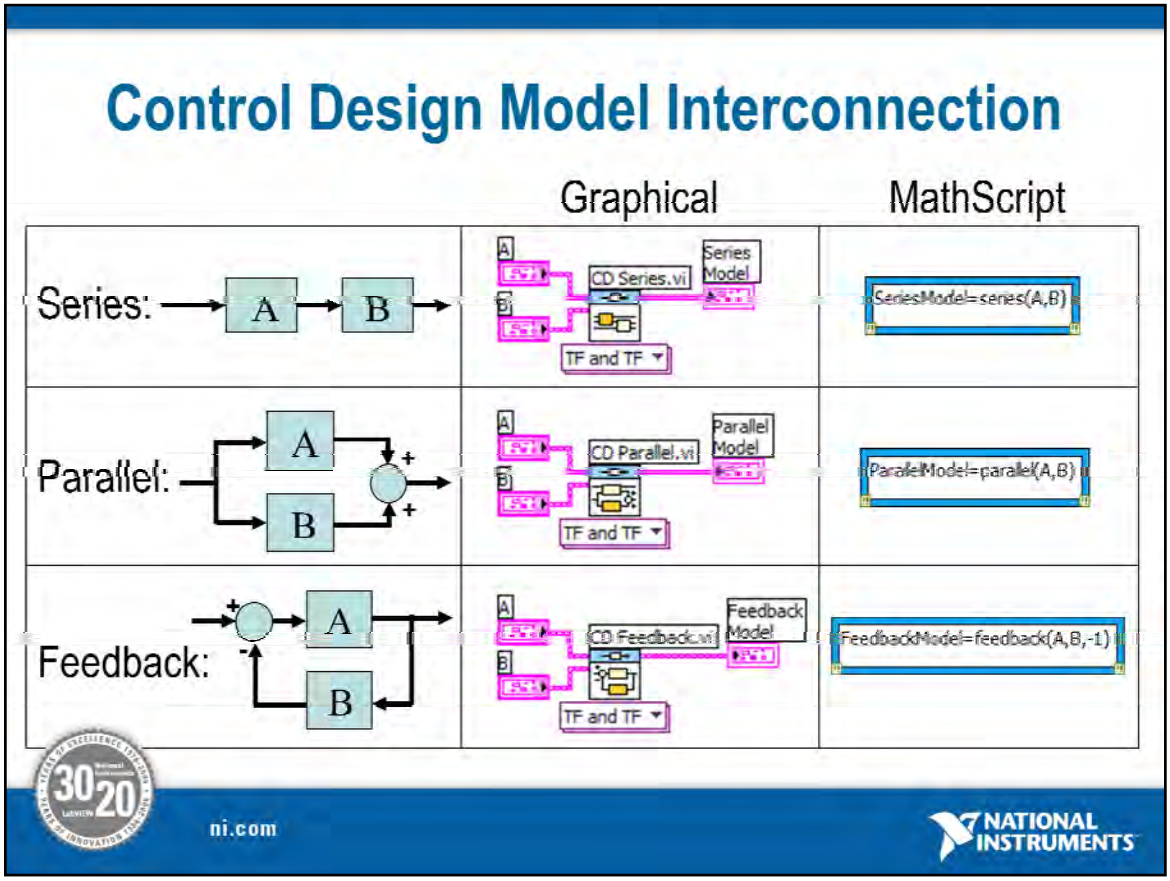
"e" represents the tracking error, which is the difference between the desired output and the actual output. This error signal (e) will be sent to the PID controller; the controller then computes the proportional, the derivative and the integral of this error signal. The output of the PID controller is now fed into the plant, which generate another output and new error is calculated. This error is once again fed back to the PID controller and process is repeated.

A proportional controller (K_p) will have the effect of reducing the rise time and steady-state error. A derivative control (K_d) will have the effect of increasing the stability of the system, reducing the overshoot, and improving the transient response and an integral control (K_i) will have the effect of eliminating the steady-state error, reduces rise time, but increases settling time.



The CD Construct PID Model VI creates a PID controller model in the form of a transfer function. This is a polymorphic VI, with options for PID Parallel, PID Academic, and PID Series models. In this course, the PID Parallel model is used. The following equation is used in the PID Parallel Model:

$U(s)/E(s) = Kc + K_i/s + K_d s/(T_f s+1)$



System models can be interconnected with the LabVIEW Control Design Toolkit either graphically or textually. Graphical interconnections can created with LabVIEW VIs from the **Model Interconnection** palette. The same interconnections can be created textually within a MathScript node with the textual Control Design functions.

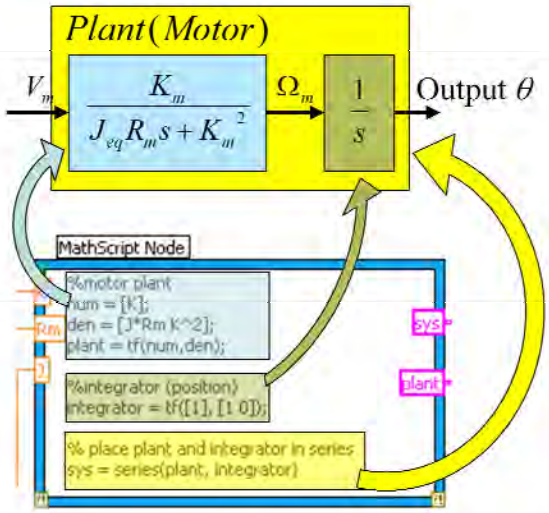
The different types of model interconnection include Series, Parallel, and Feedback connections. Each function has two input models and one output model.

- Create a Series model graphically with the CD Series VI, or textually with the Series() function.
- Create a Parallel model graphically with the CD Parallel VI, or textually with the Parallel() function.
- Create a Feedback model graphically with the CD Feedback VI, or textually with the Feedback() function.

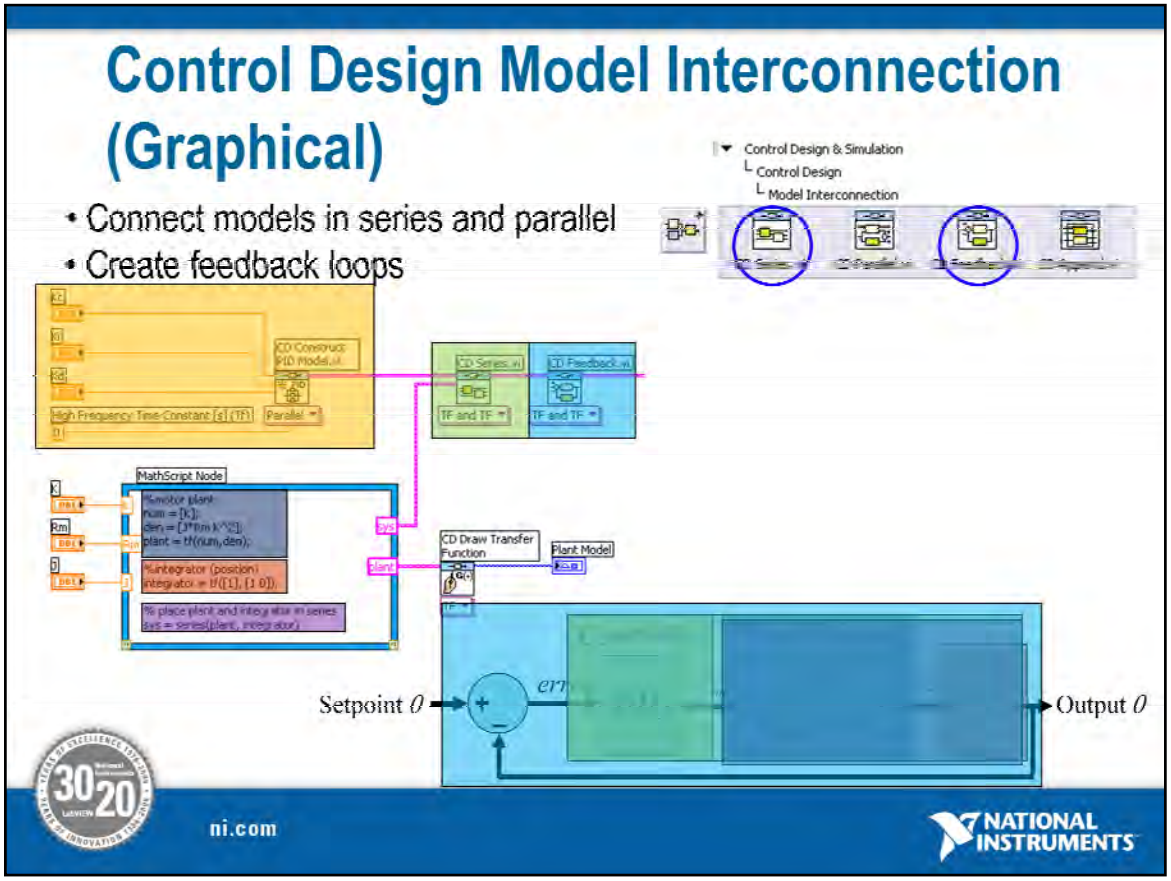
These different forms of interconnections are often used together to create complete control systems by combining controllers and plants with feedback.

Control Design Model Interconnection (MathScript)

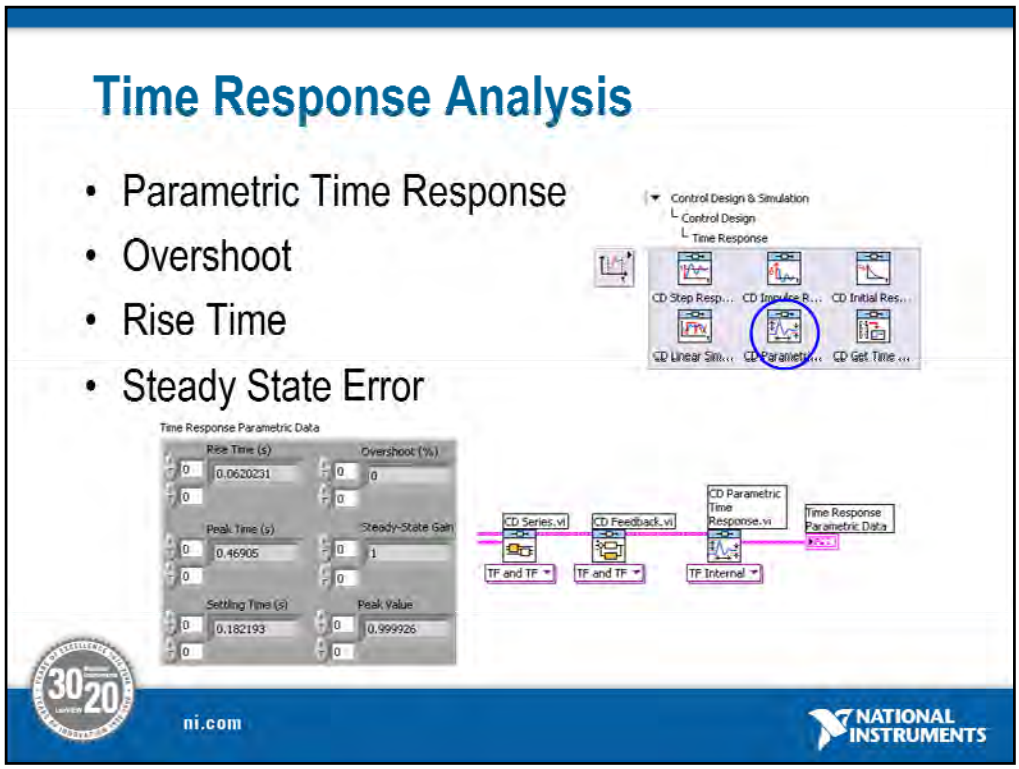
- Use MathScript commands to interconnect models
- Types of interconnection:
  - Series
  - Parallel
  - Feedback
  - Append



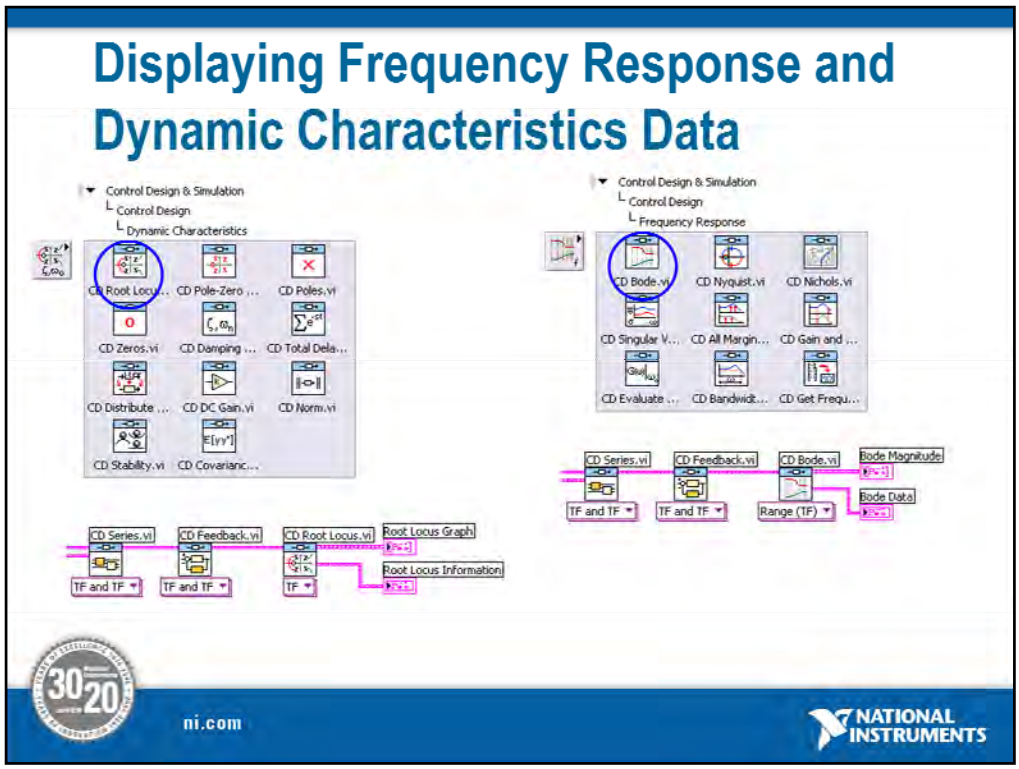
The Control Design MathScript functions include model interconnection functions. The connect functions will create an output model equivalent to the interconnected input models.



Control Design models can be interconnected graphically, as well as textually. The Model Interconnection VIs will create an output model equivalent to the interconnected input models.



The Time Response VIs can be used to create generic linear simulations and time domain plots for step inputs, impulse inputs, and initial condition responses. Results can be graphed, or parametric data about the linear simulations can be displayed.



CD Root Locus: Used to plot the Evans plot or trajectory of closed-loop poles of a single-input single-output (SISO) system as the feedback gain varies from zero to infinity.

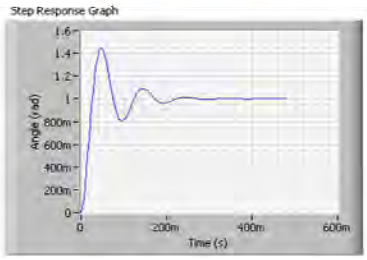
CD Bode: Produces the Bode magnitude and Bode phase plots of the system model on an XY graph.

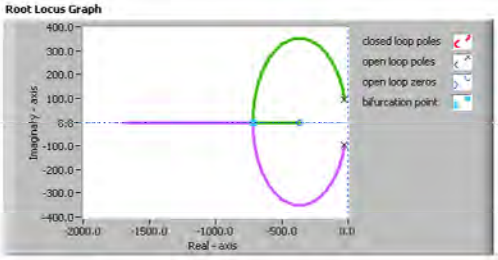
CD Nyquist: Produces the Nyquist plot of the input system for which this VI plots the imaginary part of the frequency response against its real part.

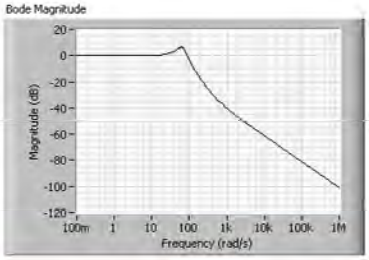
CD Pole-Zero Map: Plots the poles and zeros of a system model on an XY graph that represents a complex plane.

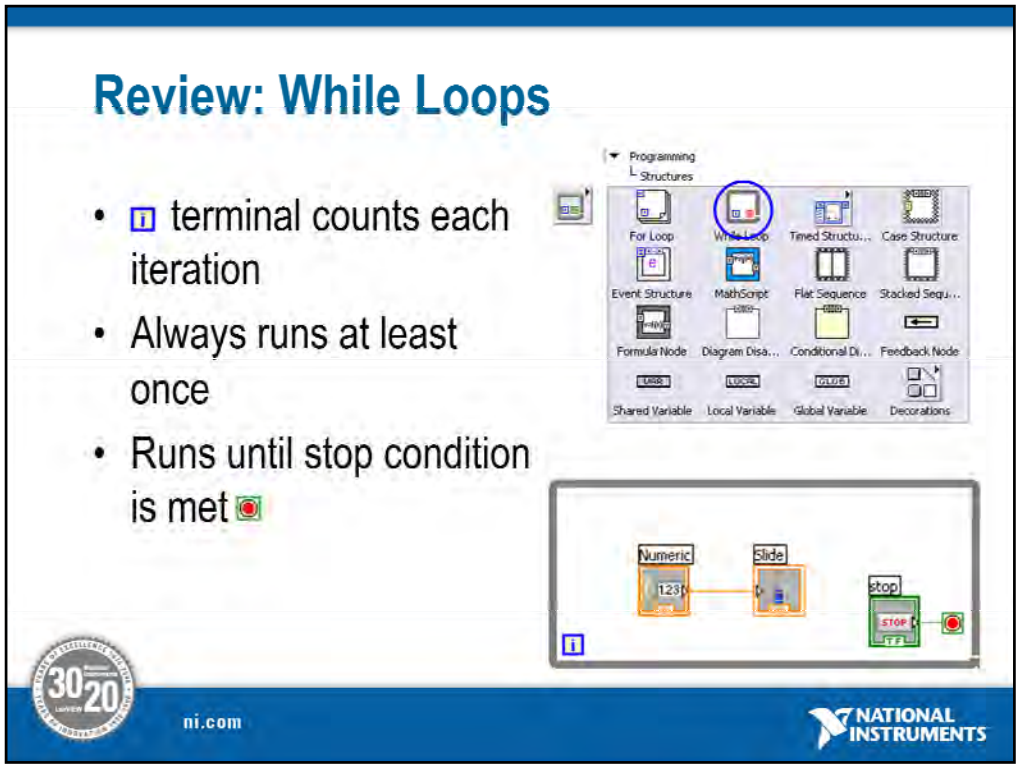
Control Design Analysis and Display

- Perform general linear simulations in the time domain
- Analyze models in the frequency domain
- Calculate dynamic properties of a model

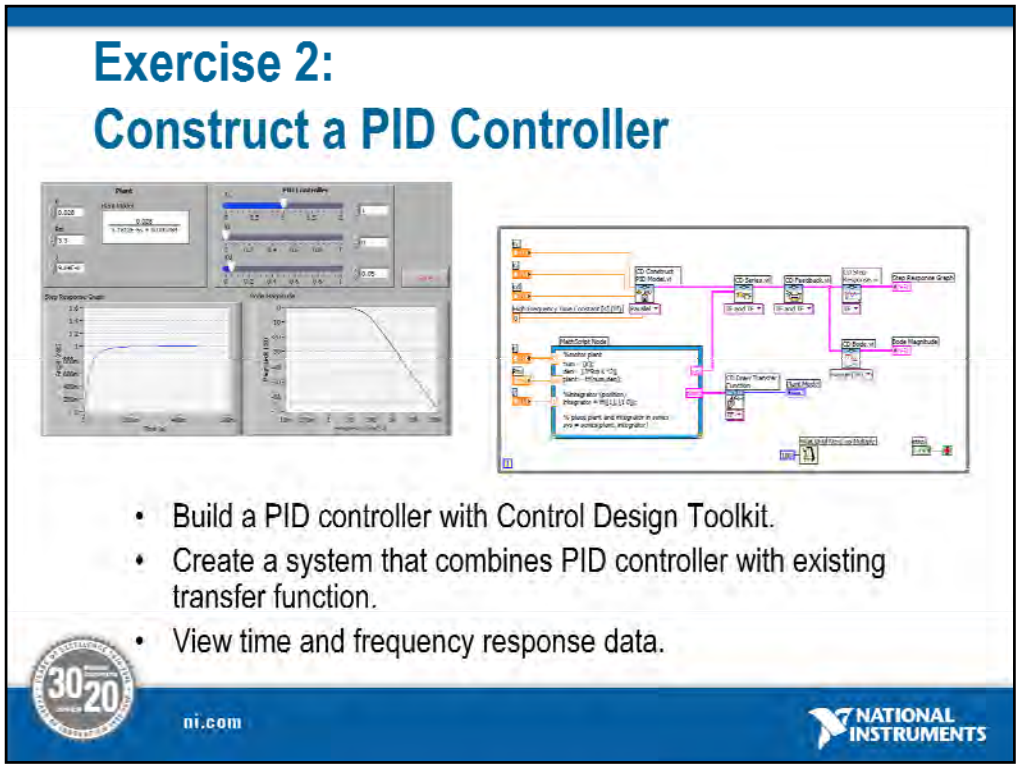








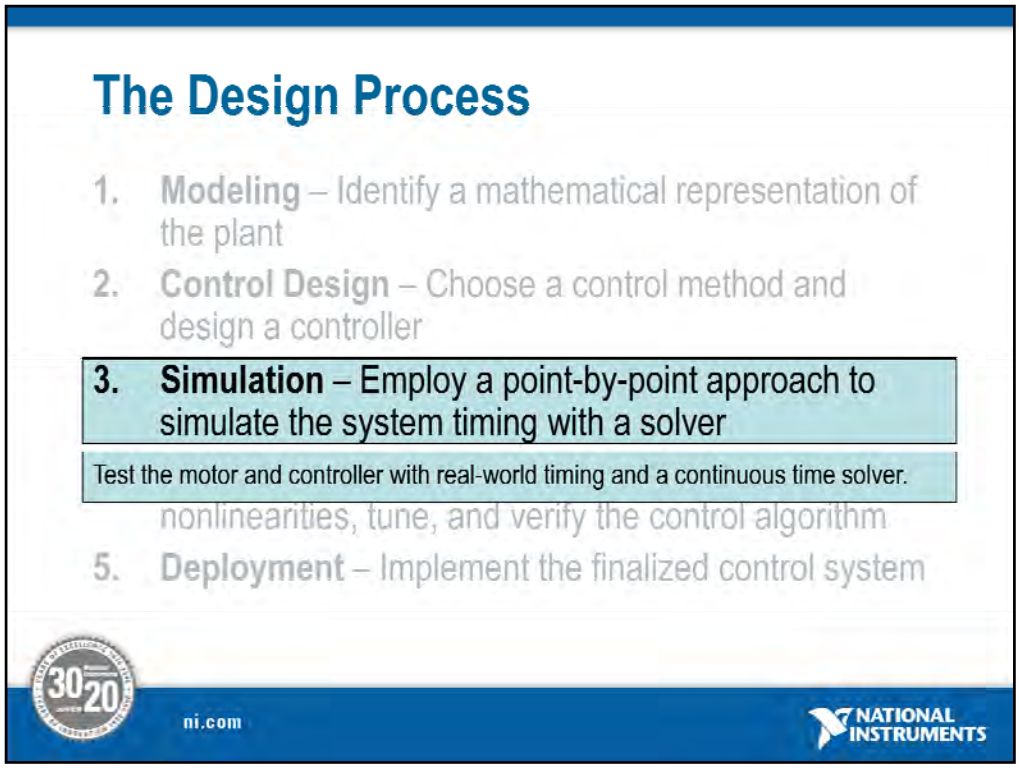
Similar to a Do Loop or a Repeat-Until Loop in text-based programming languages, a While Loop executes a sub diagram until a condition is met. The While Loop executes the sub diagram until the conditional terminal, an input terminal, receives a specific Boolean value. The default behavior and appearance of the conditional terminal is Stop If True. When a conditional terminal is Stop If True, the While Loop executes its sub diagram until the conditional terminal receives a TRUE value. The iteration terminal (an output terminal), shown at left, contains the number of completed iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns 0.



Exercise Instructions can be found at the end of PDF version of this manual or independently as “Updated Exercise Instructions.pdf”.

Step 3: Simulation

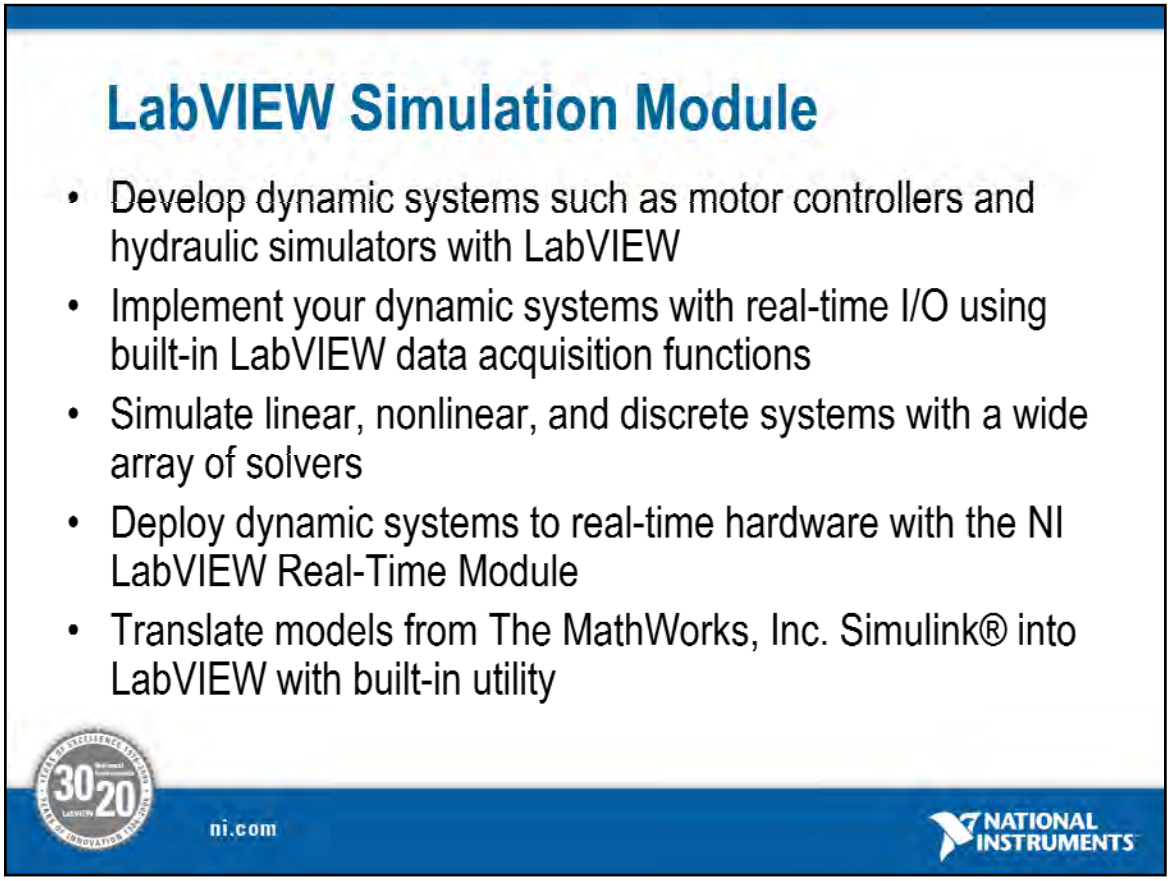
Test the controller and incorporate real-world nonlinearities



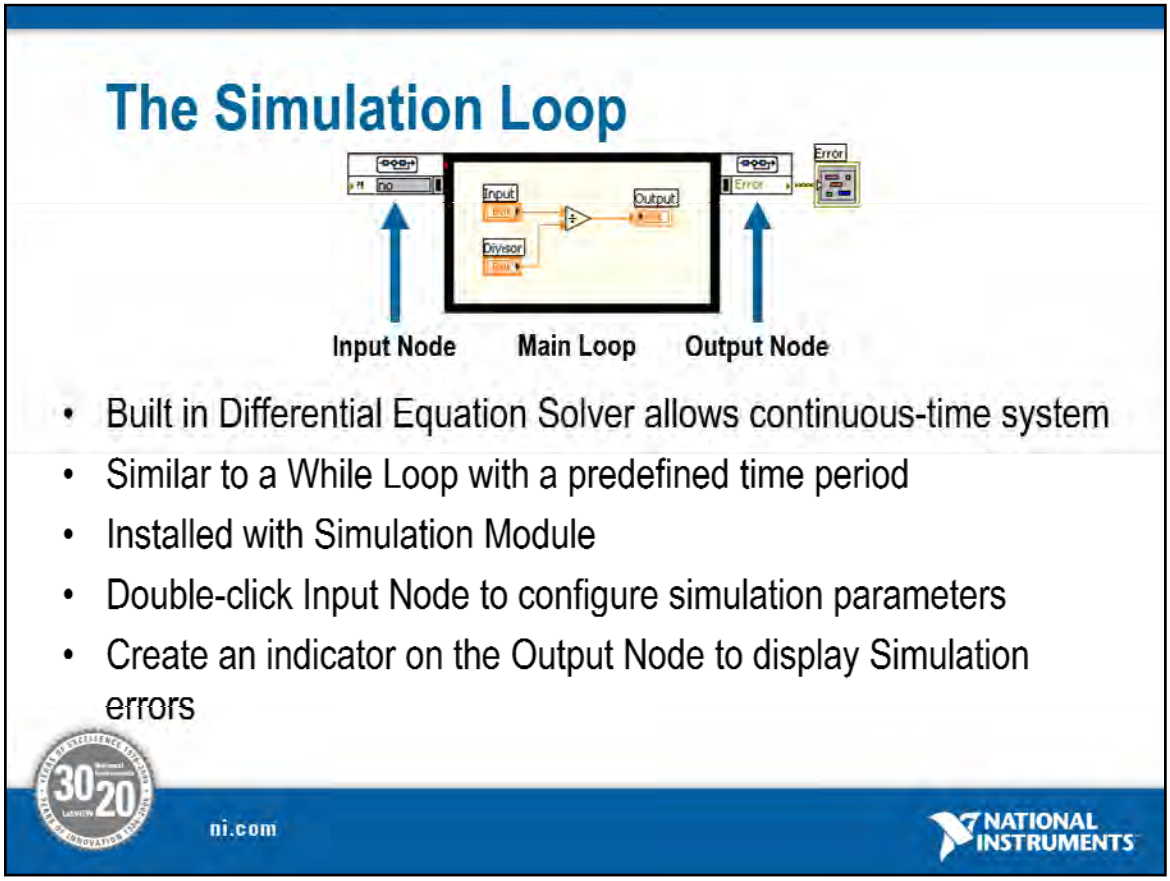
The controller can be optimized by fully understanding the plant. This understanding comes from analysis done in the LabVIEW Control Design Toolkit. Specialized graphs, such as Bode, root-locus, and Nyquist plot, builds intuition of how the plant will behave. Further analysis using these plots with an open loop system with the controller allow stability and other characteristics to be calculated. In this way, we use Control Design to optimize the parameters, in our case the gains P, I, and D. Graphs in the time domain, such as the step response, provide immediate feedback on the ideal behavior of the system, such as rise time, overshoot, settling time, and steady-state error.

We then use the Simulation Module to validate these parameters under real world constraints. One prime example is the maximum and minimum voltage and current available to be supplied to the motor. The voltage limits of the motor define the linear range of operation. Outside of that range, the motor might be damaged or become unpredictable. This is an example of nonlinear behavior, which cannot be modeled using LTI analysis found in the Control Design Toolkit. The Simulation Module allows nonlinearities to be represented in the model. Voltage limits can be modeled using the Saturation block. Other nonlinear behaviors can be written in LabVIEW, MathScript, and the LabVIEW Formula Node, and integrated into the model as a Sub VI.

The optimization process usually includes several iterations between Control Design and Simulation to achieve the desired performance of the system.



The National Instruments LabVIEW Simulation Module integrates dynamic system simulation into the LabVIEW environment. As a module, the Simulation Module adds new functionality to core LabVIEW. The simulation loop can model linear, nonlinear, discrete, and continuous plant or control systems in block diagram form. Create models from blocks such as integrator, derivative, and transfer function blocks, and then add graphs and controls to test out the models. Alternatively, you can import models developed in the NI LabVIEW Control Design Toolkit or NI LabVIEW System Identification Toolkit. The interactive nature of the LabVIEW tools allow you to modify parameters while logging the results of the simulation. Models built with the simulation node can also seamlessly download to a real-time target with the LabVIEW Real-Time Module for control prototyping and hardware-in-the-loop (HIL) simulation.

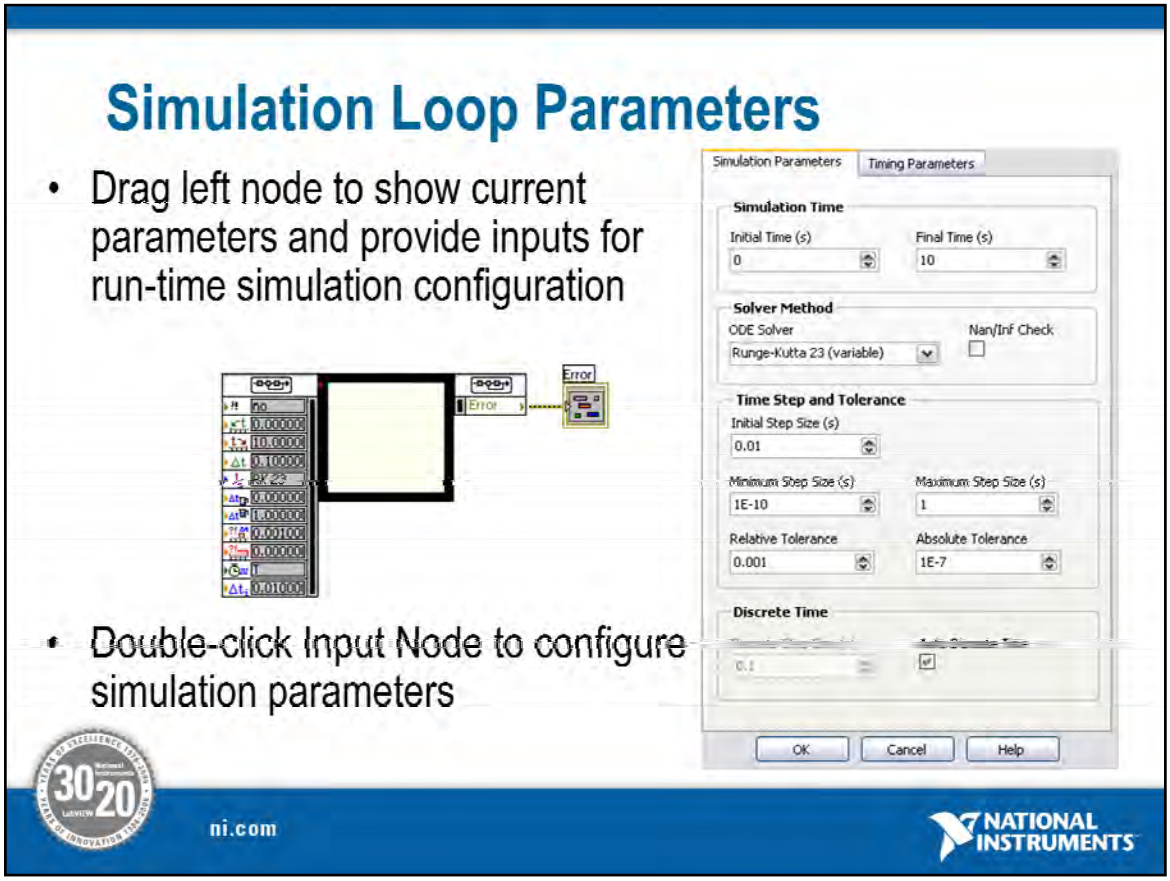


The simulation loop is the core component of the Simulation Module. It is an "upgraded" version of the while loop which integrates powerful differential equation solvers, internal or external timing features, and cross-platform capabilities. With the Simulation Module, continuous time systems can be run in the discrete digital world. The loop consists of three main parts:

Input node (Left)- allows simulation parameters to be programmatically defined. By default, these parameters are static and can be configured by double-clicking the input node. This node can be expanded by clicking on the bottom border of the node to "grab" the handle and dragging down to show additional parameters.

Main loop – The system to be simulated is placed here.

Output node (Right) – returns any errors that may have happened in the loop, such as an improper transfer function.



Configuring a Simulation

There are many parameters that can be configured for a given simulation loop.

Simulation Parameters Tab

Simulation Time – Specifies for what period of "simulation time" how long the simulation should run. This time doesn't necessarily dictate the computation time of the simulation. See Timing Parameters below.

Solver Method – Specifices what Ordinary Differential Equation ODE) solver is used to solve integral and differential type blocks in the simulation. A wide variety of solvers are available.

Time Step and Tolerance – These settings control the window of time steps used by LabVIEW. Typically, the default settings will suffice, but adjust them if necessary.

Discrete Time – While the Default Auto Discrete Time option will typically work for most simulations, you can force LabVIEW to use a specific step size here.

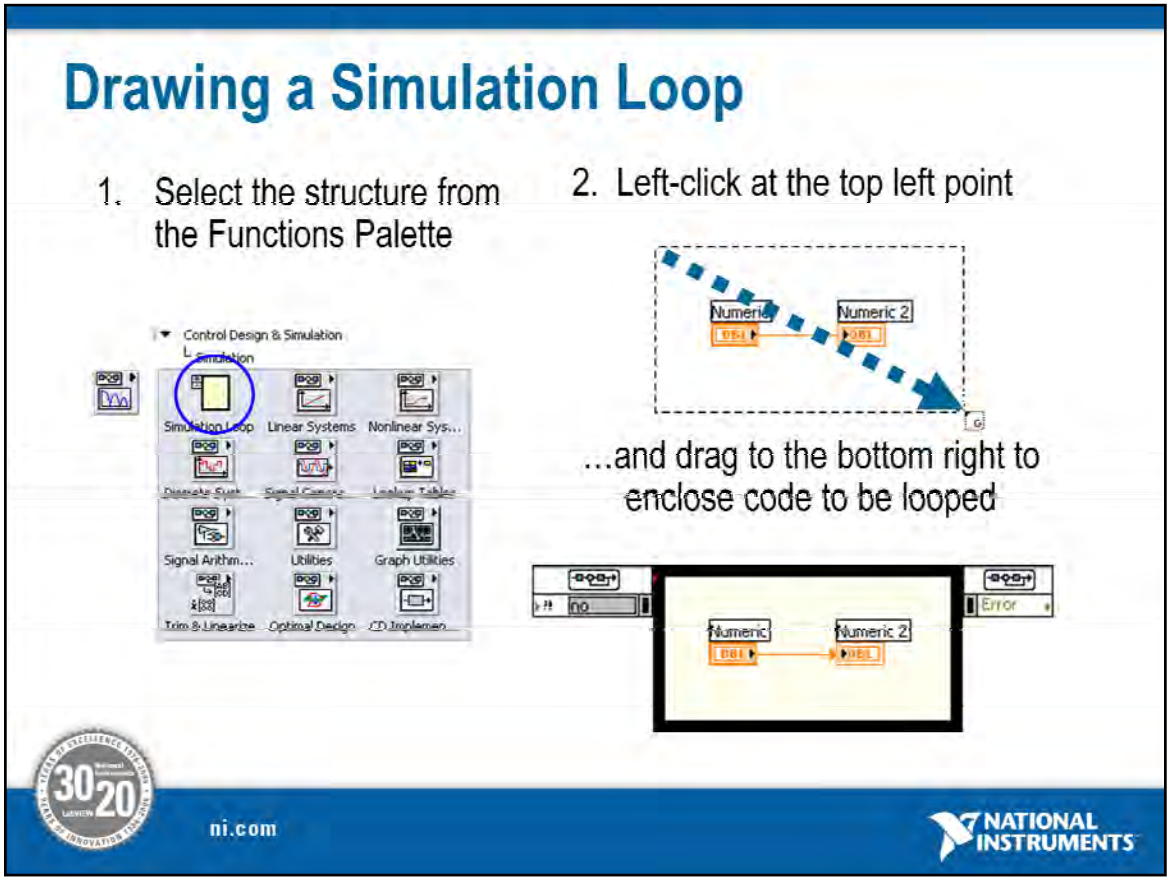
Timing Parameters Tab

Timing – The software timing option is always used when performing simulations. This will solve the equation as fast as the CPU can. To observe a simulation run in "real time", disable the software timing option and set the timing parameters below.

Loop Timing Source – When implementing a simulation in hardware, change the loop timing source to an available hardware timing. This option can be used to sync the simulation execution with an external time source, such as the operating system clock or even a Data Acquisition board's clock.

Loop Timing Parameters – These options control how the loop executes with respect to the selected timing source.

For more detailed information on these options, consult the LabVIEW help by clicking the Help button.

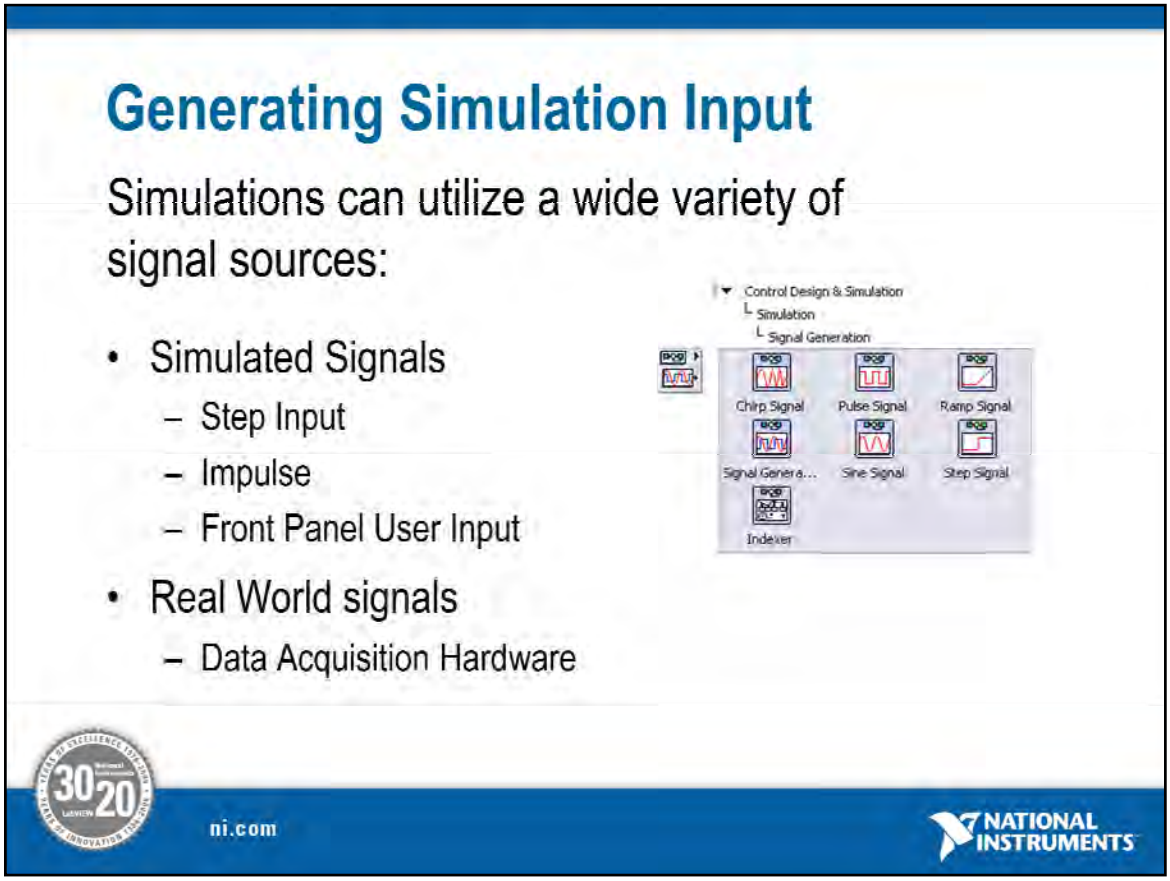


Place simulation loops in your diagram by selecting them from the Simulation Palette in the Functions palette:

When selected, the mouse cursor becomes a special pointer that you use to enclose the section of code you want to repeat.

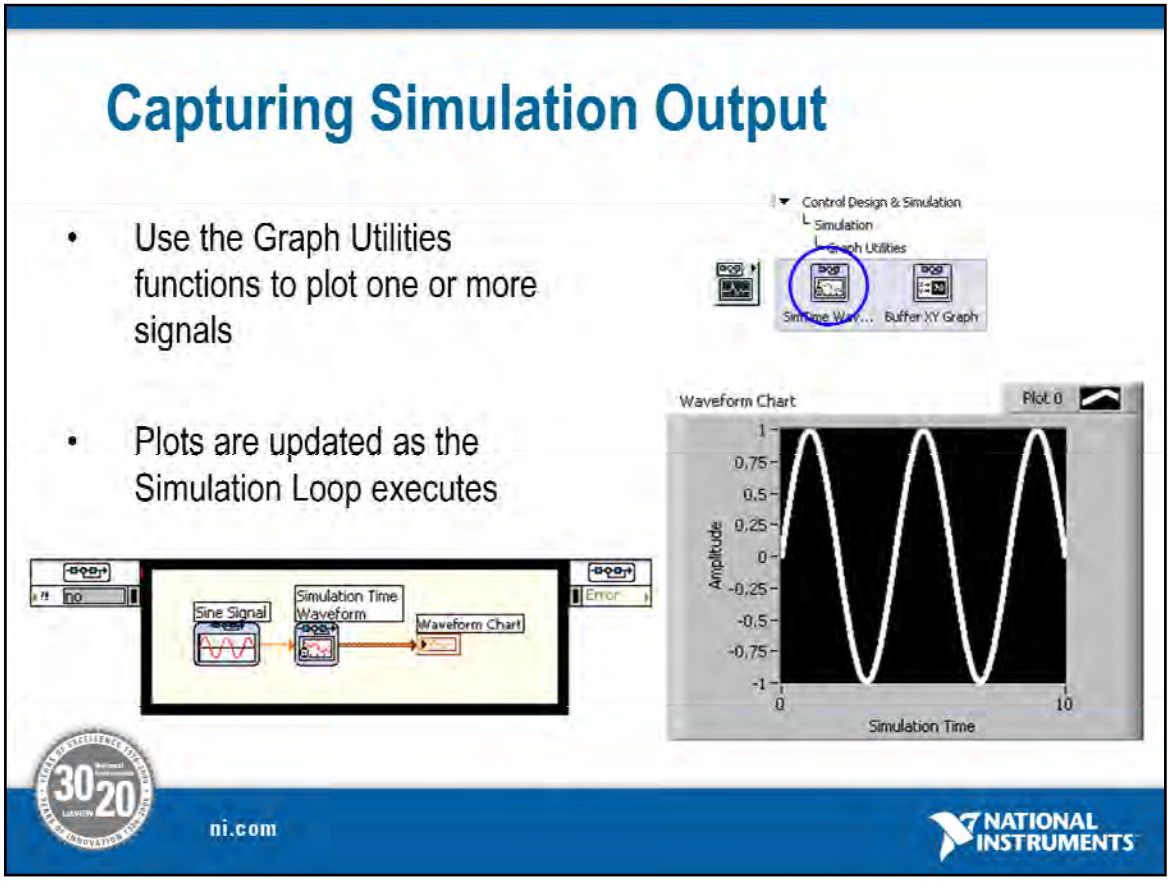
Click the mouse button to define the top-left corner, click the mouse button again at the bottom-right corner, and the Simulation Loop boundary is created around the selected code.

Drag or drop additional blocks or functions in the Simulation Loop if needed.



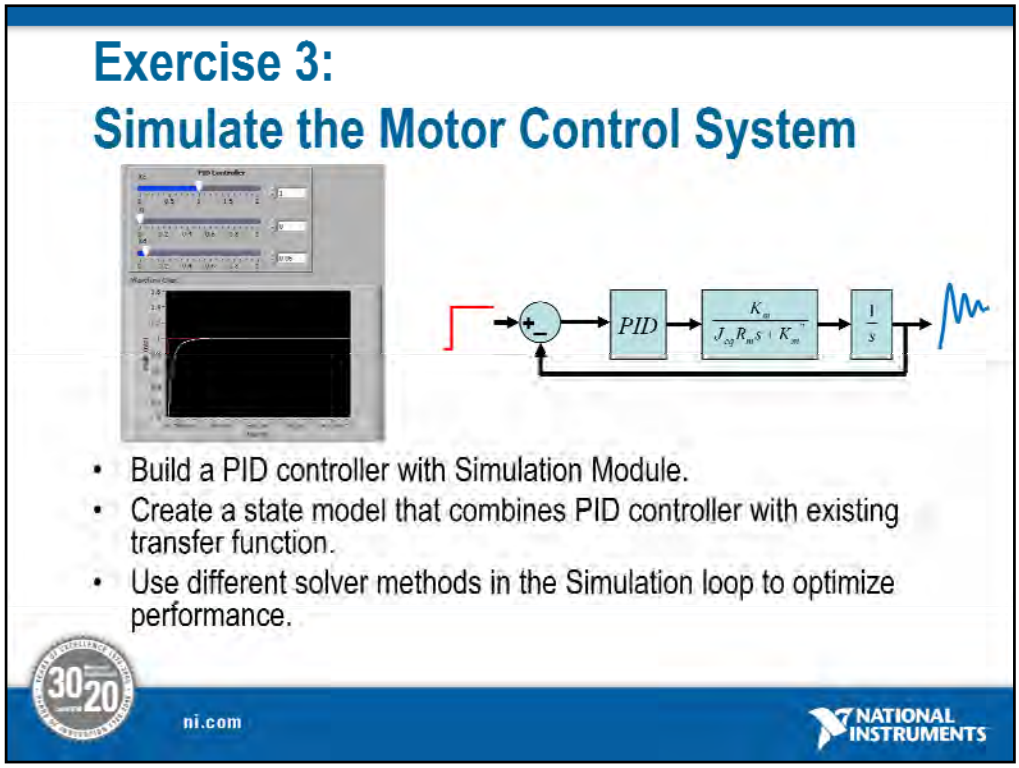
Simulations can utilize a wide variety of signals sources. Simulated signals are useful for characterizing system response and testing corner cases. Chirps provide a useful frequency sweep, and pulses provide good step response information. The indexer is useful for using a predetermined arbitrary signal, and can interpolate between samples for better resolution.

Once the simulation is confirmed, a real world signal can be substituted for the simulation signal. In this configuration, the simulation is now performing calculations based on actual data and is very good for testing the system before controlling the output.



The Simulation Time Waveform chart is a special numeric indicator that displays one or more plots. The waveform chart is located on the Functions » Control Design & Simulation » Simulation » Graph Utilities » SimTime Waveform palette. Waveform charts can display single or multiple plots. The following front panel shows an example of a multi-plot waveform chart.

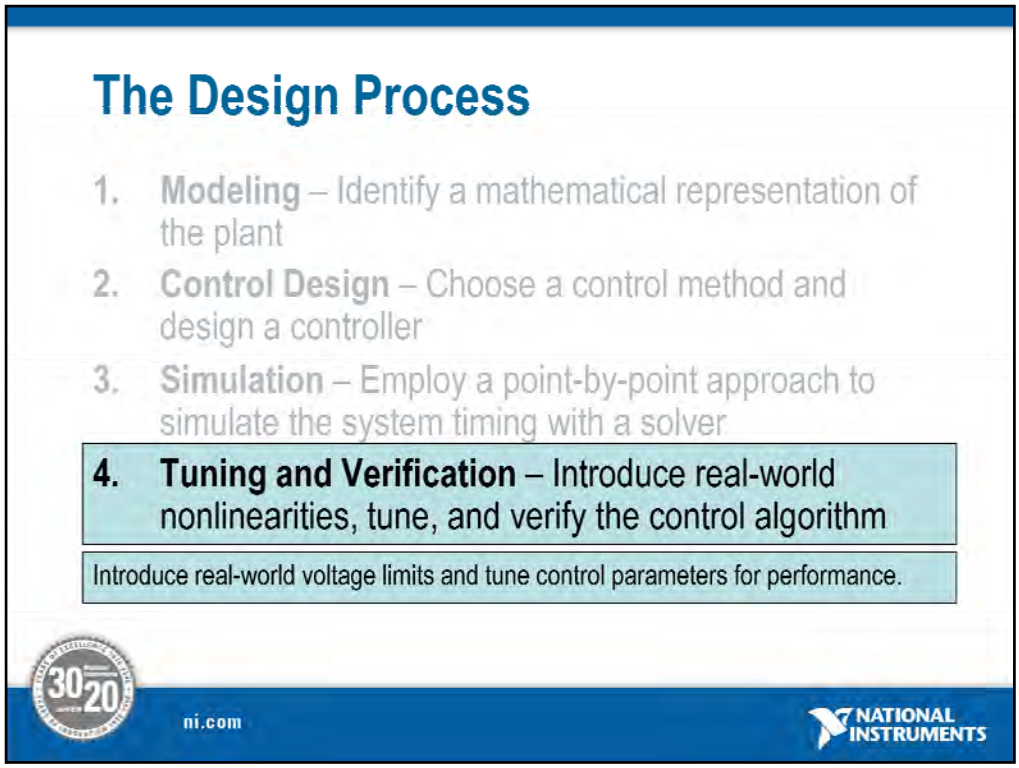
You can change the min and max values of either the x or y axis by double clicking on the value with the labeling tool and typing the new value. Similarly, you can change the label of the axis. You can also right click the plot legend and change the style, shape, and color of the trace that is displayed on the chart.



Exercise Instructions can be found at the end of PDF version of this manual or independently as “Updated Exercise Instructions.pdf”.

Step 4: Tuning and Verification

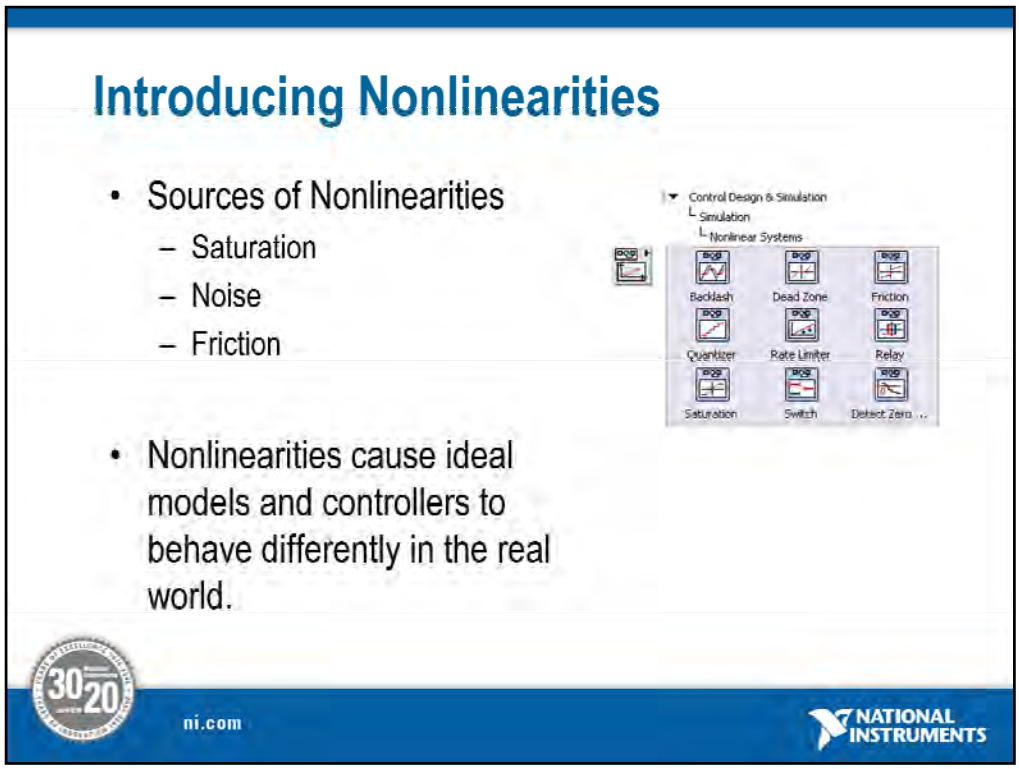
Tune the controller behavior to meet design specifications in a realistic environment



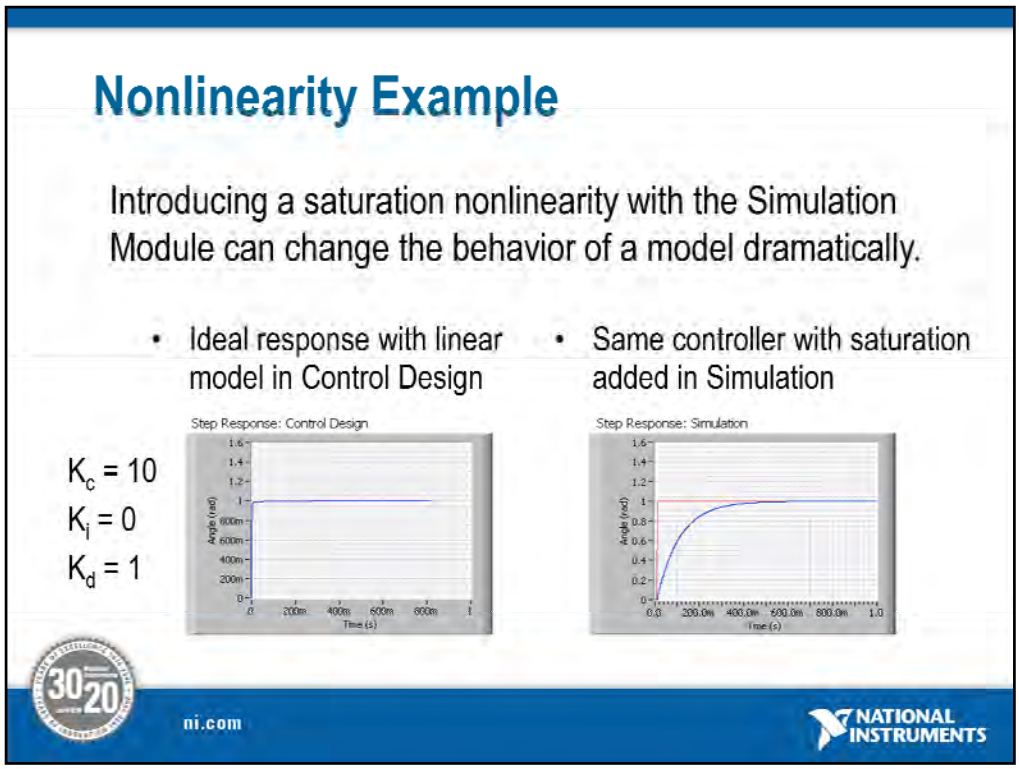
The controller can be optimized by fully understanding the plant. This understanding comes from analysis done in the LabVIEW Control Design Toolkit. Specialized graphs, such as Bode, root-locus, and Nyquist plot, builds intuition of how the plant will behave. Further analysis using these plots with an open loop system with the controller allow stability and other characteristics to be calculated. In this way, we use Control Design to optimize the parameters, in our case the gains P, I, and D. Graphs in the time domain, such as the step response, provide immediate feedback on the ideal behavior of the system, such as rise time, overshoot, settling time, and steady-state error.

We then use the Simulation Module to validate these parameters under real world constraints. One prime example is the maximum and minimum voltage and current available to be supplied to the motor. The voltage limits of the motor define the linear range of operation. Outside of that range, the motor might be damaged or become unpredictable. This is an example of nonlinear behavior, which cannot be modeled using LTI analysis found in the Control Design Toolkit. The Simulation Module allows nonlinearities to be represented in the model. Voltage limits can be modeled using the Saturation block. Other nonlinear behaviors can be written in LabVIEW, MathScript, and the LabVIEW Formula Node, and integrated into the model as a Sub VI.

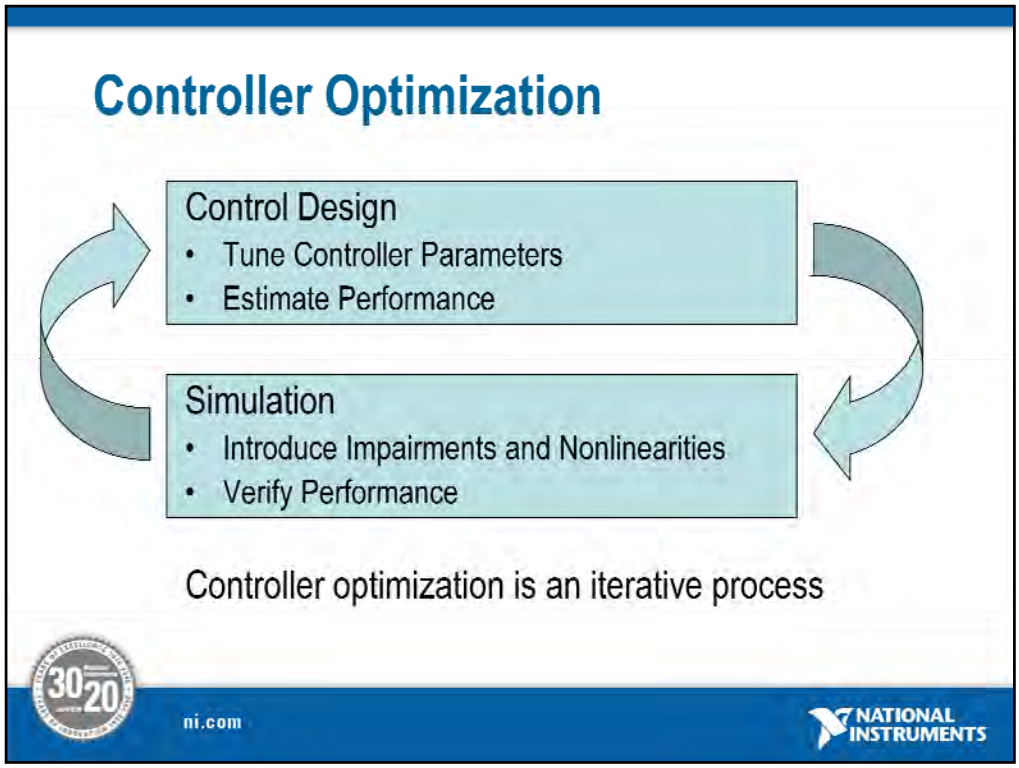
The optimization process usually includes several iterations between Control Design and Simulation to achieve the desired performance of the system.



Nonlinearities cause ideal models and controllers to behave different than expected in the real world. Sources of nonlinearities include saturation, noise, and friction, which cannot be accounted for with a linear system model. The Simulation Module can analyze nonlinear models, allowing us to simulate non-ideal behavior before deployment.



In this example, a PID controller model is created with the Control Design toolkit. The PID parameters are tuned so that the step response of the system is very close to ideal. The controller model is then used in the Simulation Module with the same PID parameters, where a saturation nonlinearity is introduced. The nonlinearity represents the maximum and minimum voltage (24V, -24V) that can be supplied to the motor. The introduction of this saturation changes the behavior of the system so that it has a far less ideal behavior. This real world constraint of saturation must now be accounted for, and the PID controller parameters need to be readjusted to give a better step response while staying within the constraints dictated by the nonlinearity.



The controller can be optimized by fully understanding the plant. This understanding comes from analysis done in the LabVIEW Control Design Toolkit. Specialized graphs, such as Bode, root-locus, and Nyquist plots build intuition of how the plant will behave. Further analysis using these plots with an open loop system with the controller allow stability and other characteristics to be calculated. In this way, we use Control Design to optimize the parameters, in our case the gains P, I, and D. Graphs in the time domain, such as the step response, provide immediate feedback on the ideal behavior of the system, such as rise time, overshoot, settling time, and steady-state error.

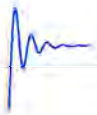
We then use the Simulation Module to validate these parameters under real world constraints. One prime example is the maximum and minimum voltage and current available to be supplied to the motor. The voltage limits of the motor define the linear range of operation. Outside of that range, the motor might be damaged or become unpredictable. This is an example of nonlinear behavior, which cannot be modeled using LTI (Linear Time Invariant) analysis found in the Control Design Toolkit. The Simulation Module allows nonlinearities to be represented in the model. Voltage limits can be modeled using the Saturation block. Other nonlinear behaviors can be written in LabVIEW, MathScritp, and the LabVIEW Formula Node, and integrated into the model as a Sub VI.

The optimization process usually includes several iterations between Control Design and Simulation to achieve the desired performance of the system.

Tuning your PID Controller

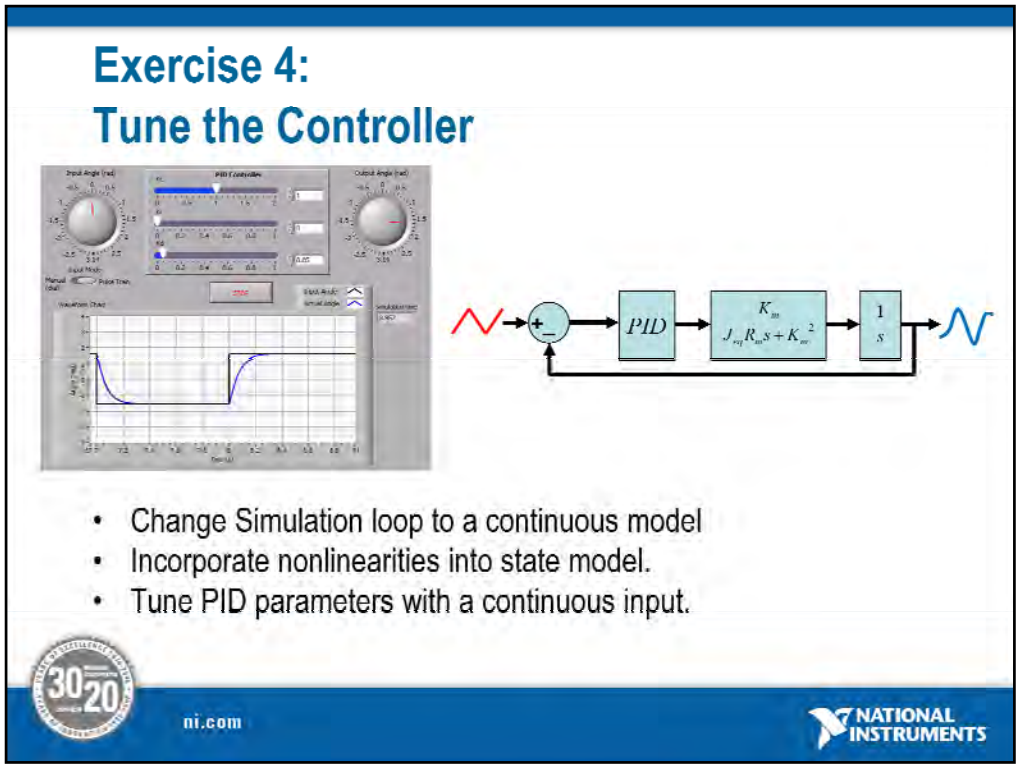
Tune PID controller design using the step response

- Begin with Gains set at: $K_c = 1$, $K_i = 0$, and $K_d = 0$
- Increase Proportional Gain (K_c) to get desired rise time



- Increase Derivative Gain (K_d) to reduce overshoot and settling time

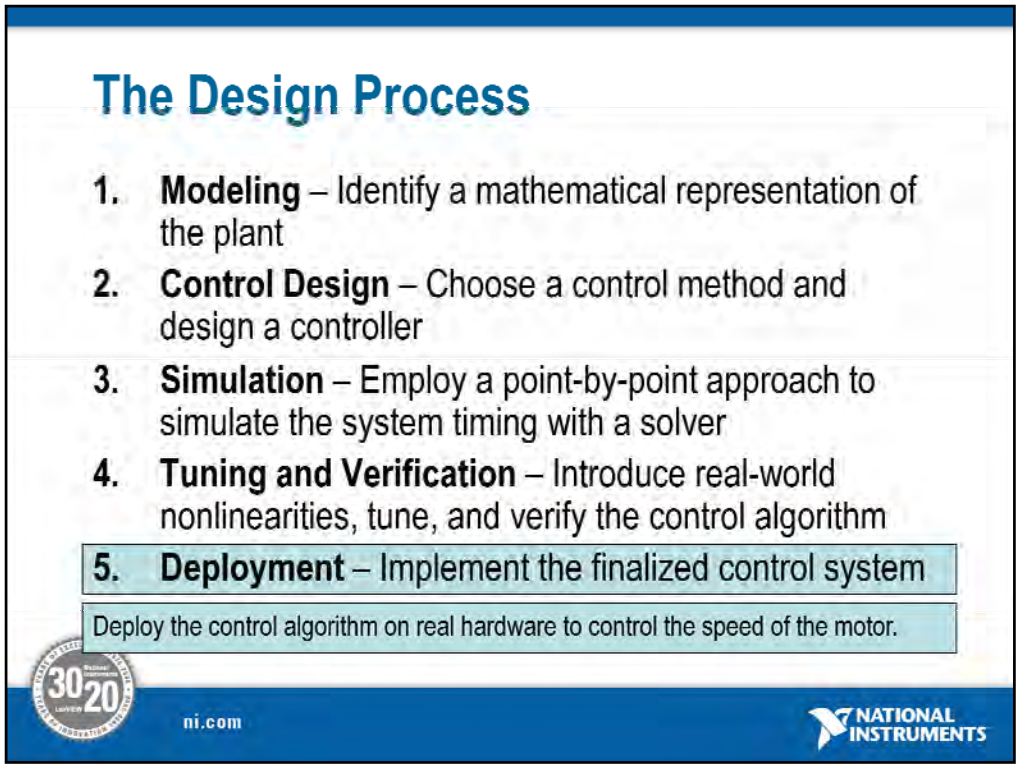
- Increase Integral Gain (K_i) to reduce steady-state error if necessary



Exercise Instructions can be found at the end of PDF version of this manual or independently as “Updated Exercise Instructions.pdf”.

Step 5: Deployment

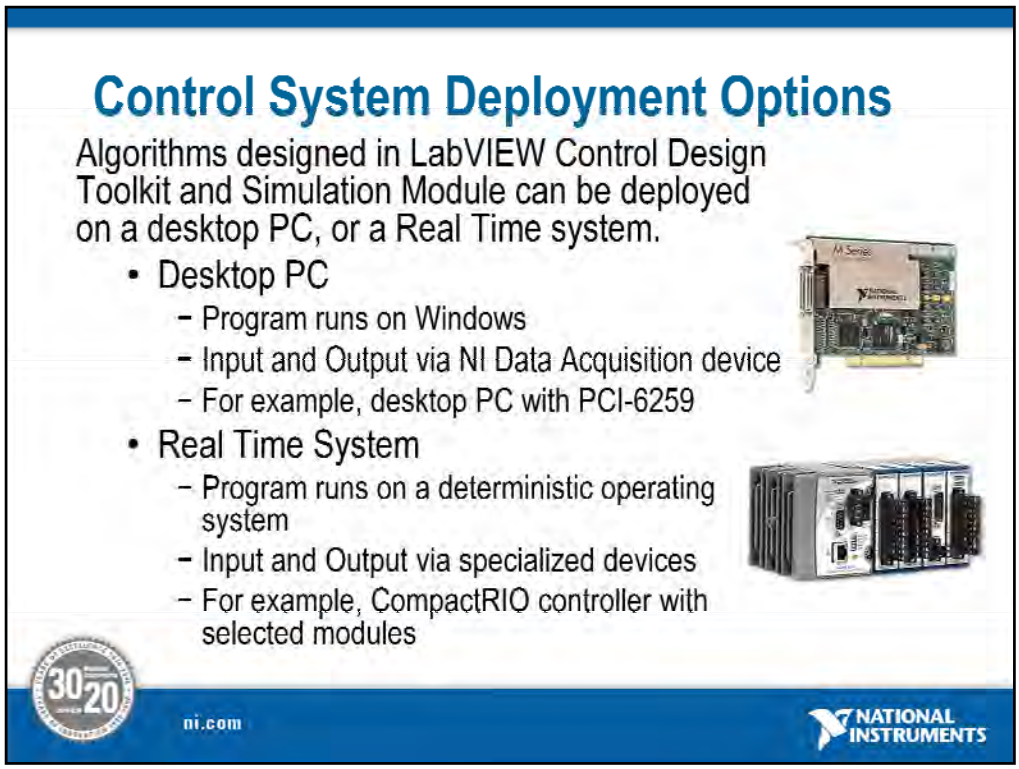
Implement the finalized control system



The controller can be optimized by fully understanding the plant. This understanding comes from analysis done in the LabVIEW Control Design Toolkit. Specialized graphs, such as Bode, root-locus, and Nyquist plot, builds intuition of how the plant will behave. Further analysis using these plots with an open loop system with the controller allow stability and other characteristics to be calculated. In this way, we use Control Design to optimize the parameters, in our case the gains P, I, and D. Graphs in the time domain, such as the step response, provide immediate feedback on the ideal behavior of the system, such as rise time, overshoot, settling time, and steady-state error.

We then use the Simulation Module to validate these parameters under real world constraints. One prime example is the maximum and minimum voltage and current available to be supplied to the motor. The voltage limits of the motor define the linear range of operation. Outside of that range, the motor might be damaged or become unpredictable. This is an example of nonlinear behavior, which cannot be modeled using LTI analysis found in the Control Design Toolkit. The Simulation Module allows nonlinearities to be represented in the model. Voltage limits can be modeled using the Saturation block. Other nonlinear behaviors can be written in LabVIEW, MathScript, and the LabVIEW Formula Node, and integrated into the model as a Sub VI.

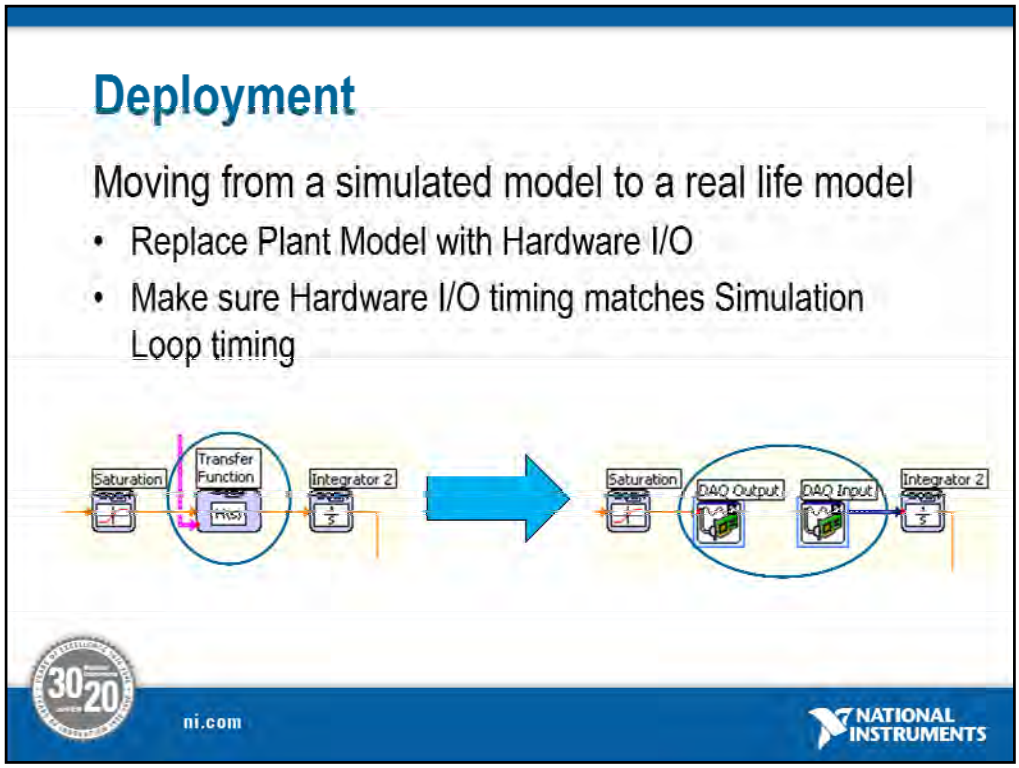
The optimization process usually includes several iterations between Control Design and Simulation to achieve the desired performance of the system.



Once a control system is designed with the Control Design Toolkit and the Simulation Module, the algorithm can be deployed to control a real physical system. The deployed algorithm can be run on a desktop PC or a Real Time system.

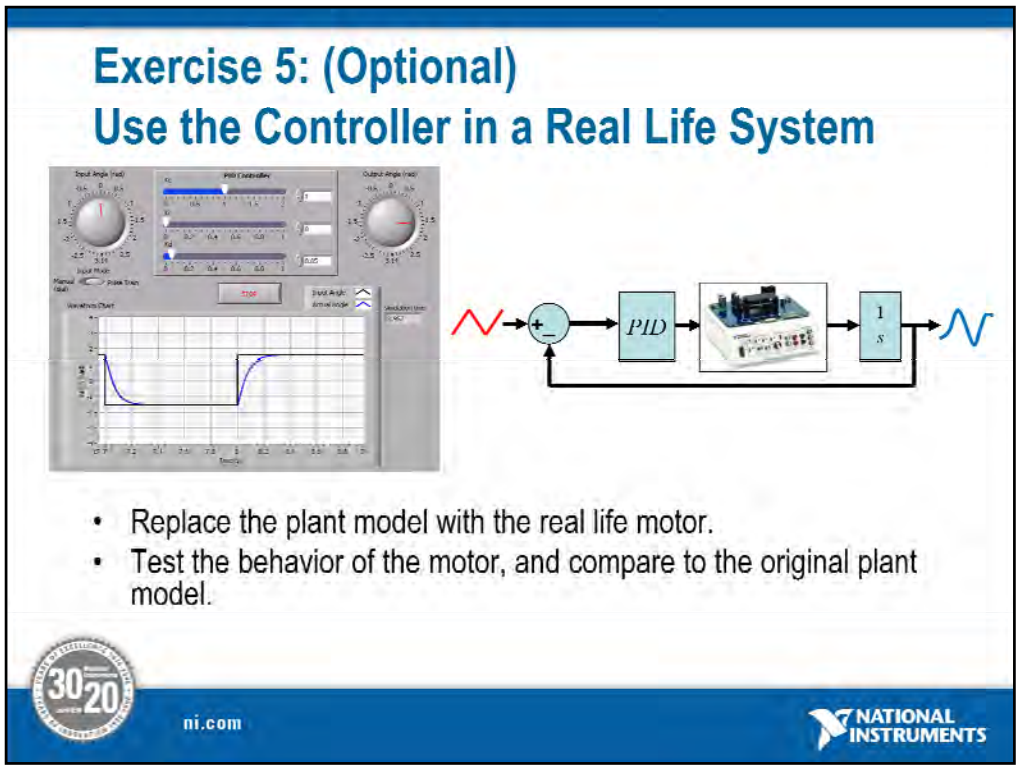
Running a control system on a desktop PC involves simply replacing the plant model with hardware input and output functions. The system can then be run in LabVIEW just as with the simulated system. One example would be to replace the transfer function representing the plant in the Simulation loop with NI-DAQmx input and output VIs. The DAQmx VIs would control an NI Data Acquisition device like the PCI-6259.

One limitation of running a control system on a desktop PC is the non-deterministic nature of desktop operating systems. Background processes can interrupt the timing of the algorithm, which can cause inaccurate results. National Instruments provides Real Time standalone platforms, such as the CompactRIO, that ensure predictable timing of onboard processes. This provides superior performance with applications that require high speed and high accuracy control.



The LabVIEW Simulation Module can be used to control real life systems as well as simulated models. To migrate from simulated control to real life control, the plant model can be replaced with hardware input and output functions. In this case, we will replace the transfer function representing the motor with DAQmx input and output VIs that control the actual motor.

Timing is an important consideration when deploying with the Simulation Module. Since the Simulation loop uses a built in ODE solver with time steps, it is important to set the Simulation Parameters and the Timing Parameters of the loop to have the same time step. Data acquisition tasks typically use timing parameters, so it is also important to equate the Simulation loop timing with the data acquisition timing.



Exercise 5:

Note: This exercise requires the NI ELVIS, Quanser QNET DC Motor, and an NI PCI DAQ device. NI-DAQmx 8.3 or later must also be installed.

1. Open `Exercise 4.vi`.
2. Save the VI as `Exercise 5.vi`.
3. Go to the block diagram and select the Transfer Function VI inside the Simulation loop. Select **Edit» Create Simulation Subsystem**. This will create a subsystem that contains the Transfer Function VI, which can be edited in a separate VI.
4. Right-click the new subsystem, and select “Open Subsystem” from the shortcut menu. This will open the subsystem containing the Transfer Function.
5. Save the new subsystem as `Ex 5 Plant.vi`. Go to the block diagram.
6. Delete the Transfer Function VI. Remove any broken wires (<CTRL-B>).
7. Place a DAQ Assistant on the block diagram: **Measurement I/O » NI-DAQmx » DAQ Assistant**.

8. Configure the DAQ Assistant for scaled analog output:
    - ❑ In the configuration window, select **Analog Output » Voltage**. Select analog output channel 0 of the NI PCI DAQ device. Click **Finish**.
    - ❑ In the Voltage Output Setup section, select **Create New** in the **Custom Scaling** pull down menu.
    - ❑ Select **Linear**, then name the scale "voltageout". Click **Finish**.
    - ❑ Set the **Slope** to 2.2. Click **OK**.
    - ❑ Set the **Signal Output Range MAX** and **MIN** values to 22 and -22, respectively.
    - ❑ In the Timing Settings section, select **1 Sample (On Demand)** for **Generation Mode**.
    - ❑ Click **OK** to exit the DAQ Assistant.
    - ❑ Wire the numeric control of `Ex 5 Plant.vi` to the **data** input of the DAQ Assistant.
9. Place a second DAQ Assistant on the block diagram: **Measurement I/O » NI-DAQmx » DAQ Assistant**.
10. Configure the DAQ Assistant for scaled analog input:
    - ❑ In the configuration window, select **Analog Input » Voltage**. Select analog input channel 4 of the NI PCI DAQ device. Click **Finish**.
    - ❑ In the Voltage Input Setup section, select **Create New** in the **Custom Scaling** pull down menu.
    - ❑ Select **Linear**, then name the scale "velocity". Click **Finish**.
    - ❑ Set the **Slope** to 109.9. Click **OK**.
    - ❑ Set the **Signal Output Range MAX** and **MIN** values to 1099 and -1099, respectively.
    - ❑ In the Timing Settings section, select **1 Sample (On Demand)** for **Acquisition Mode**.
    - ❑ Click **OK** to exit the DAQ Assistant.
    - ❑ Wire the numeric indicator of `Ex 5 Plant.vi` to the **data** output of the DAQ Assistant.
11. Save both `Exercise 5.vi` and `Ex 5 Plant.vi`.
12. Make sure that the NI ELVIS, Quanser DC Motor, and NI DAQ device are connected and powered on. The NI ELVIS should be set to "Bypass" mode.
13. Run the VI. The DC motor should respond similarly to the simulated motor in the previous exercise.

End of Exercise 5

Where Can I Learn More?

We have only begun to explore the many opportunities for control and simulation within LabVIEW. Learn more by visiting the following links:

System Identification Toolkit:
http://sine.ni.com/nips/cds/view/p/lang/en/nid/13853

Control Design Toolkit:
http://sine.ni.com/nips/cds/view/p/lang/en/nid/13854

Simulation Module:
http://sine.ni.com/nips/cds/view/p/lang/en/nid/13852

LabVIEW Real-Time Module:
http://www.ni.com/realtime

Data Acquisition and Control Hardware:
http://www.ni.com/dataacquisition

CompactRIO Real-Time Platform:
http://www.ni.com/compactrio

Quanser Control Challenges

Implement and evaluate feedback strategies such as PID, LQC, adaptive or nonlinear controllers with Quanser's linear, rotary, specialty, mechatronics and custom control experiments. Our systems are uniquely modular and use the same base components that can be multi-purposed throughout a wide range of experiment configurations. So by adding and removing various modules you extend the functionality of your investment and more importantly create a greater variety of experiments and challenges to explore – no other solution provider comes close to providing this kind of flexibility.

Quanser Engineering Trainers For NI ELVIS (QNET)

The Quanser Engineering Trainers For NI ELVIS (QNET) Series is a new cost-effective line of controls education experiments that considerably increases the value of your investment in NI ELVIS & LabVIEW software. The boards are easily connected to the NI ELVIS workstation to facilitate a diverse range of controls education experiments, thereby extending the functionality of the NI platform.

LabVIEW ECP Real-Time Control Option

In addition to the fully integrated ECP Executive® programs, National Instruments LabVIEW and NI data acquisition hardware can be used to develop and deploy real-time control algorithms to control select ECP electromechanical plants. The LabVIEW® environment allows for the design and simulation of control algorithms with seamless real-time deployment and execution. This approach allows for multiple levels of learning by allowing students to simply modify controller parameters on a graphical user interface or to design and create their own controller. It allows students to gain familiarity with the LabVIEW software and NI hardware that is widely used in industrial control applications.

Additional Resources

- NI Academic Controls Web
  - http://www.ni.com/academic/controls
- LabVIEW Student Edition DVD with Control Design and Simulation
  - http://www.academicsuperstore.com/ search: LabVIEW
  - Part Number: 752412

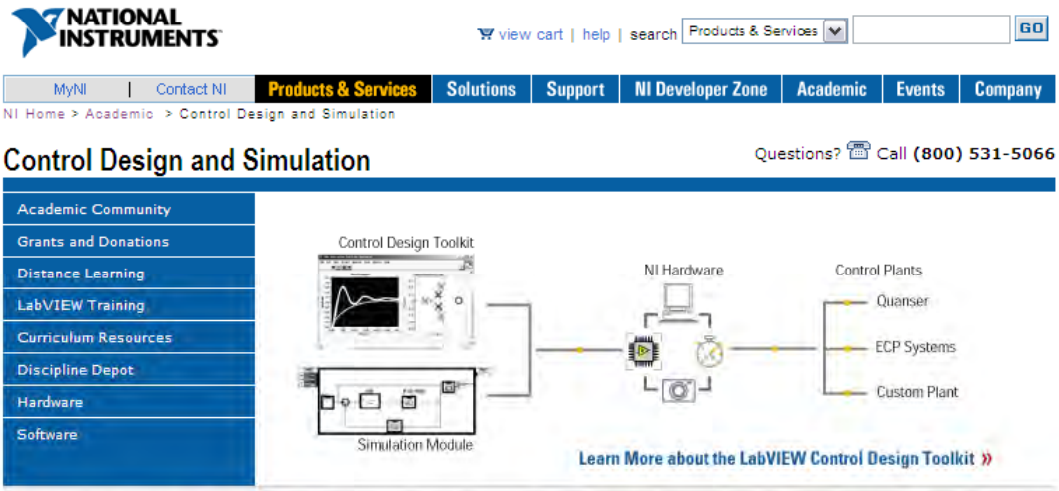
- Connexions: Full LabVIEW Introductory Course
  - www.cnx.rice.edu
  - Or search for "LabVIEW basics"

- LabVIEW Certification
  - LabVIEW Fundamentals Exam (free on www.ni.com/academic)
  - Certified LabVIEW Associate Developer Exam (industry recognized certification)

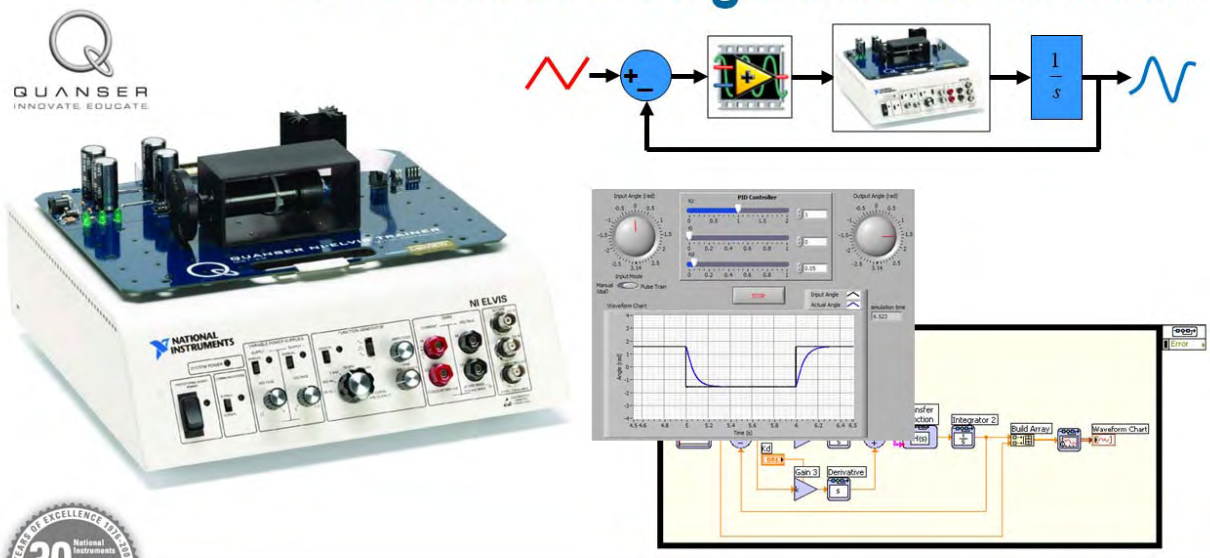
ni.com

NATIONAL INSTRUMENTS

http://www.ni.com/academic/controls



Introduction to LabVIEW in 3 Hours for Control Design and Simulation

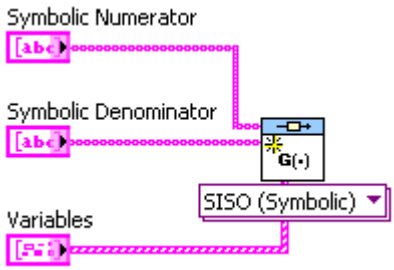


Updated Exercises for LabVIEW 8.5

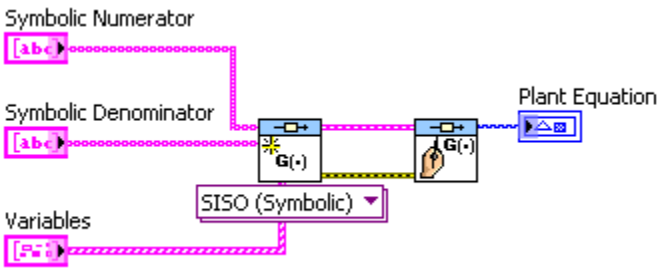
Updated 4/2008

Exercise 1a:

1. Build the VI block diagram
   - ☐ Open a blank VI from the "Getting Started" screen. Save it as `Exercise_1a.vi`.
   - ☐ Switch to the block diagram of the VI (<Ctrl+E>).
   - ☐ Open the Functions Palette by right-clicking anywhere in the block diagram. Dock the palette by clicking the icon at the top-left corner of the palette window.
   - ☐ Place the `CD Construct Transfer Function Model.vi` on the block diagram. From the functions palette, navigate to **Control Design & Simulation » Control Design » Model Construction » CD Construct Transfer Function Model.vi**, and place it on the block diagram. From the drop-down menu select **Single-Input Single-Output (Symbolic)**.
   - ☐ Right-click the **Symbolic Numerator** input terminal of the `CD Construct Transfer Function Model.vi`, and select **Create » Control** from the shortcut menu.
   - ☐ Right-click the **Symbolic Denominator** input terminal of the `CD Construct Transfer Function Model.vi`, and select **Create » Control** from the shortcut menu.
   - ☐ Right-click the **Variables** input terminal of the `CD Construct Transfer Function Model.vi`, and select **Create » Control** from the shortcut menu.
   - ☐ The block diagram should look like this:

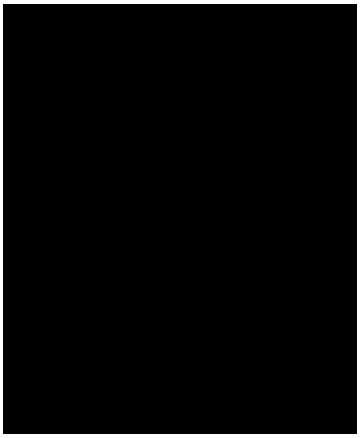


   - ☐ Place the `CD Draw Transfer Function Equation.vi` on the block diagram. From the functions palette, navigate to **Control Design & Simulation » Control Design » Model Construction » CD Draw Transfer Function Equation.vi**, and place it on the block diagram.
   - ☐ Wire the **Transfer Function Model** output terminal of the `CD Construct Transfer Function Model.vi` to the **Transfer Function Model** input terminal of the `CD Draw Transfer Function Equation.vi`.
   - ☐ Right-click the **Equation** output terminal of the `CD Draw Transfer Function Equation.vi`, and select **Create » Indicator** from the shortcut menu.
   - ☐ Triple-click the label of the picture indicator, and rename it `Plant Equation`.
   - ☐ The block diagram should look like this:



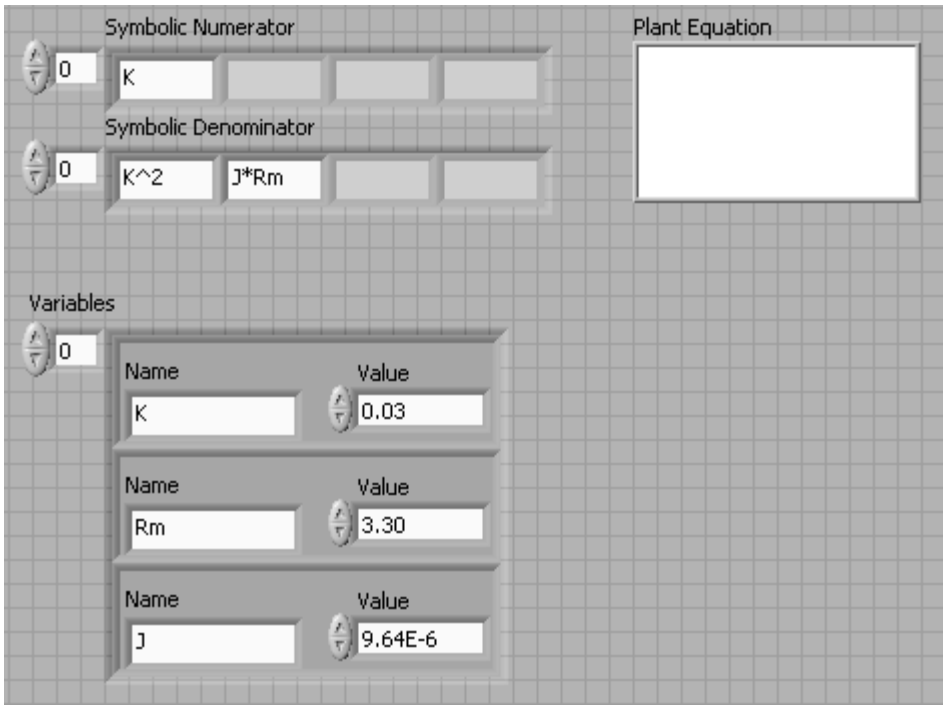
☐ Save the VI (<Ctrl+S>).

2. Run the VI
   - ☐ Switch to the front panel of the VI (<Ctrl+E>).
   - ☐ Right-click the `Value` numeric control inside the `Variables` control and select **Display Format…**, then select **Automatic Formatting**. This will allow the control to display small numbers properly.
   - ☐ On the front panel, set the values of the `Symbolic Numerator`, `Symbolic Denominator`, and `Variables` controls according to the following table:



Note: Symbolic variables are case sensitive, so make sure to use consistent capitalization.

   - ☐ Set the front panel control values to default. Select **Edit » Make Current Values Default**.
   - ☐ The front panel should look like this:



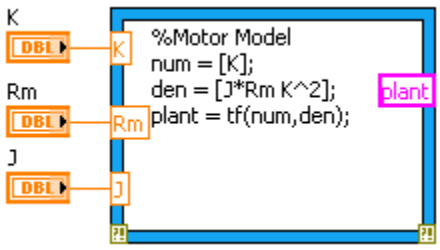
- ☐ Run the VI (<Ctrl+R>). Observe the transfer function drawn in the `Plant Equation` indicator. This transfer function will be used in future exercises.
- ☐ Save the VI (<Ctrl+S>).

Exercise 1b:

1. Build the VI block diagram
   - ☐ Open a blank VI from the "Getting Started" screen. Save it as `Exercise_1b.vi`.
   - ☐ Switch to the block diagram of the VI (<Ctrl+E>).
   - ☐ Open the Functions Palette by right-clicking anywhere in the block diagram. Dock the palette by clicking the icon at the top-left corner of the palette window.
   - ☐ Place a MathScript Node on the block diagram. From the functions palette, navigate to **Programming » Structures » MathScript Node**, and draw a rectangle on the block diagram to define the node.
   - ☐ Enter the following code inside the MathScript Node:

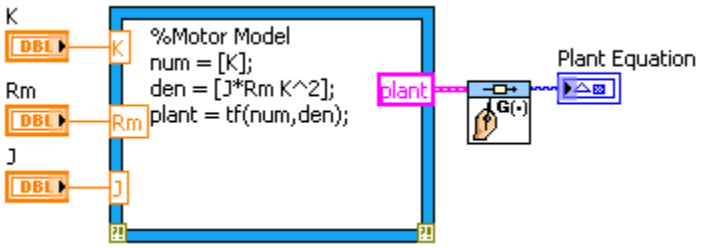
```
%Motor Model
num = [K];
den = [J*Rm K^2];
plant = tf(num,den);
```

   - ☐ Right-click on the left border of the MathScript Node and select **Add Input**. Enter K as the input. Right-click on the input terminal of the K variable and select **Create » Control** from the shortcut menu.
   - ☐ Right-click on the left border of the MathScript Node and select **Add Input**. Enter `Rm` as the input. Right-click on the input terminal of the `Rm` variable and select **Create » Control** from the shortcut menu.
   - ☐ Right-click on the left border of the MathScript Node and select **Add Input**. Enter `J` as the input. Right-click on the input terminal of the `J` variable and select **Create » Control** from the shortcut menu.
   - ☐ Right-click on the right border of the MathScript Node and select **Add Output**. Enter `plant` as the output. Right-click on the output terminal of the `plant` variable and select **Choose Data Type » Add-ons » TF Object**.
   - ☐ The block diagram should look like this:



   - ☐ Place the `CD Draw Transfer Function Equation.vi` on the block diagram. From the functions palette, navigate to **Control Design & Simulation » Control Design » Model Construction » CD Draw Transfer Function Equation.vi**, and place it on the block diagram.
   - ☐ Wire the output terminal of the `plant` variable to **Transfer Function Model** input terminal of the `CD Draw Transfer Function Equation.vi`.
   - ☐ Right-click the **Equation** output terminal of the `CD Draw Transfer Function Equation.vi`, and select **Create » Indicator** from the shortcut menu.

- ☐ Double-click the label of the picture indicator, and rename it `Plant Equation`.
- ☐ The block diagram should look like this:



- ☐ Save the VI (<Ctrl+S>).

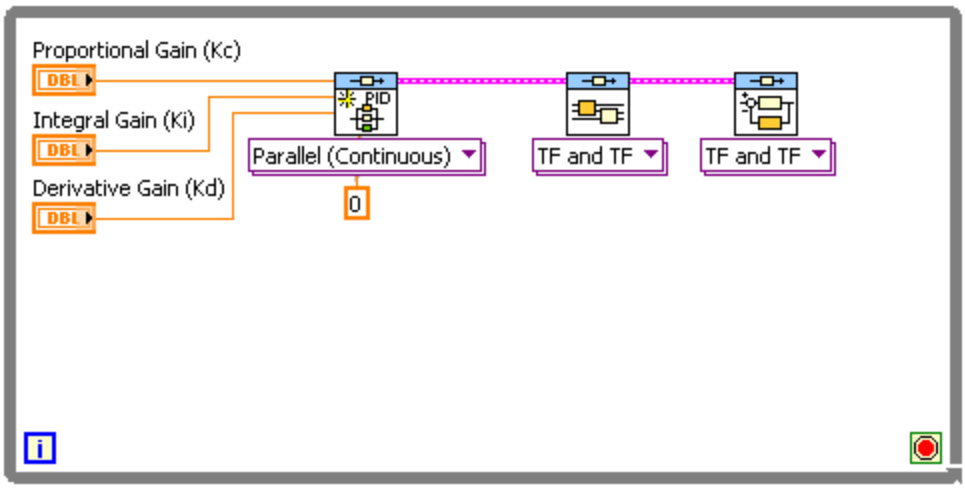
2. Run the VI
   - ☐ Switch to the front panel of the VI (<Ctrl+E>).
   - ☐ Set the values of the front panel controls according to the following table:

K	0.028
Rm	3.3
J	9.64E-06

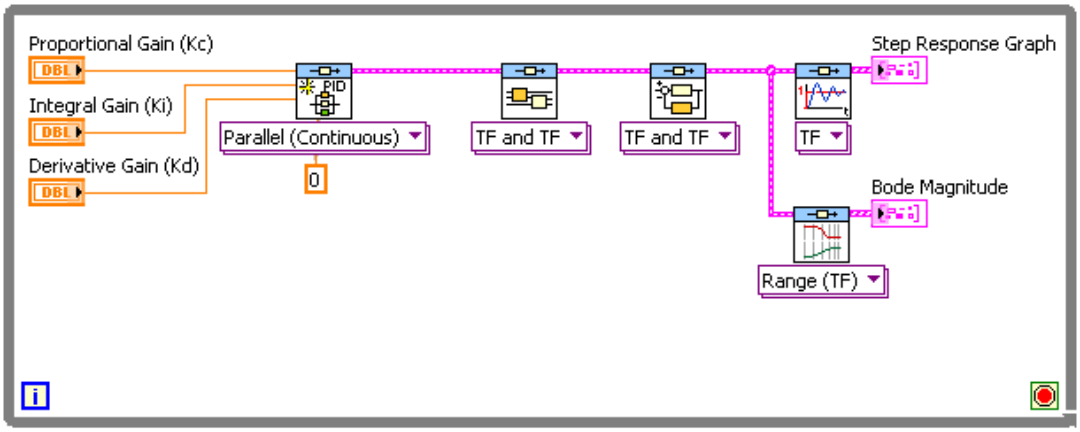
   - ☐ Set the front panel control values to default. Select **Edit » Make Current Values Default**.
   - ☐ Run the VI (<Ctrl+R>). Observe the transfer function drawn in the `Plant Equation` indicator. This transfer function will be used in future exercises.
   - ☐ Save the VI (<Ctrl+S>).

Exercise 2:

1. Build the VI block diagram
    - ☐ Open a blank VI from the "Getting Started" screen. Save it as `Exercise_2.vi`.
    - ☐ Switch to the block diagram of the VI (<Ctrl+E>).
    - ☐ Open the Functions Palette by right-clicking anywhere in the block diagram. Dock the palette by clicking the icon at the top-left corner of the palette window.
    - ☐ Place a **While Loop** on the block diagram. From the functions palette, navigate to **Programming » Structures » While Loop**. Draw a rectangle on the block diagram to define the loop.
    - ☐ Place the **CD Construct PID Model.vi** on the block diagram. From the functions palette, navigate to **Control Design & Simulation » Control Design » Model Construction » CD Construct PID Model.vi**, and place it on the block diagram. From the drop-down menu select **PID Parallel**.
    - ☐ Right-click on the **Kc** input terminal of the `CD Construct PID Model.vi` and select **Create » Control** from the shortcut menu.
    - ☐ Right-click on the **Ki** input terminal of the `CD Construct PID Model.vi` and select **Create » Control** from the shortcut menu.
    - ☐ Right-click on the **Kd** input terminal of the `CD Construct PID Model.vi` and select **Create » Control** from the shortcut menu.
    - ☐ Right-click on the **High Frequency Time Constant [s] (Tf)** input terminal of the `CD Construct PID Model.vi` and select **Create » Constant** from the shortcut menu.
    - ☐ Place the **CD Series.vi** on the block diagram. From the functions palette, navigate to **Control Design & Simulation » Control Design » Model Interconnection » CD Series.vi**, and place it on the block diagram. From the drop-down menu select **Transfer Function and Transfer Function**.
    - ☐ Wire the **Transfer Function Model** output terminal of the `CD Construct PID Model.vi` to the **Model 1** Input terminal of the `CD Series.vi`.
    - ☐ Place the **CD Feedback.vi** on the block diagram. From the functions palette, navigate to **Control Design & Simulation » Control Design » Model Interconnection » CD Feedback.vi**, and place it on the block diagram. From the drop-down menu select **Transfer Function and Transfer Function**.
    - ☐ Wire the **Series Model** output terminal of the `CD Series.vi` to the **Model 1** input terminal of the `CD Feedback.vi`.
    - ☐ The block diagram should look like this:



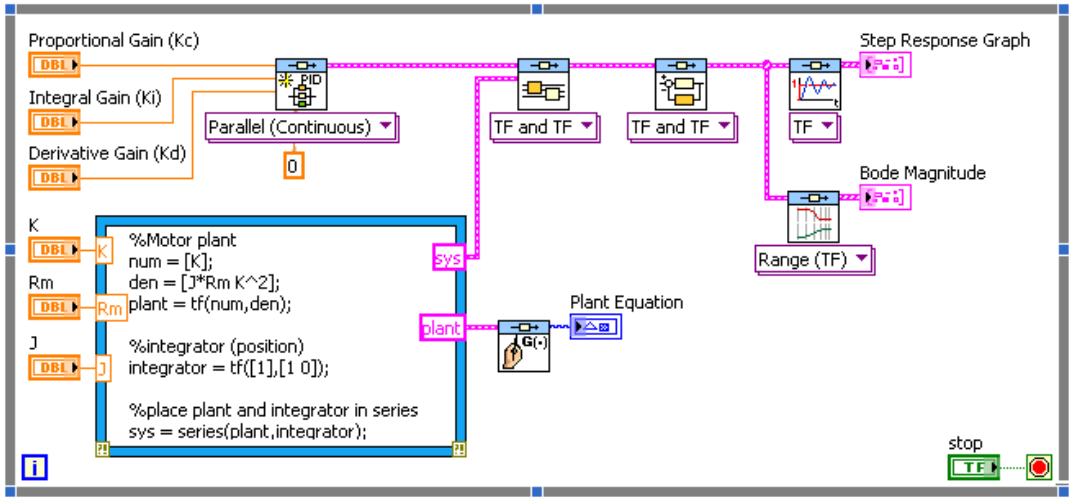
- ☐ Place the **CD Step Response.vi** on the block diagram. From the functions palette, navigate to **Control Design & Simulation » Control Design » Time Response » CD Step Response.vi**, and place it on the block diagram. From the drop-down menu select **Transfer Function**.
- ☐ Wire the **Closed Loop Model** output terminal of the `CD Feedback.vi` to the **Transfer Function Model** input terminal of the `CD Step Response.vi`.
- ☐ Right-click the **Step Response Graph** output terminal of the `CD Step Response.vi` and select **Create » Indicator** from the shortcut menu.
- ☐ Place the **CD Bode.vi** on the block diagram. From the functions palette, navigate to **Control Design & Simulation » Control Design » Frequency Response » CD Bode.vi**, and place it on the block diagram. From the drop-down menu select **Frequency Range » Transfer Function**.
- ☐ Wire the **Closed Loop Model** output terminal of the `CD Feedback.vi` to the **Transfer Function Model** input terminal of the `CD Bode.vi`.
- ☐ Right-click the **Bode Magnitude** output terminal of the `CD Bode.vi` and select **Create » Indicator** from the shortcut menu.
- ☐ The block diagram should look like this:



- ☐ Place a **MathScript Node** on the block diagram. From the functions palette, navigate to **Programming » Structures » MathScript Node**, and draw a rectangle on the block diagram to define the node.
- ☐ Enter the following code inside the MathScript Node:

```
%Motor plant
num = [K];
den = [J*Rm K^2];
plant = tf(num,den);
%integrator (position)
integrator = tf([1],[1 0]);
%place plant and integrator in series
sys = series(plant,integrator);
```

- ☐ Right-click on the left border of the MathScript Node and select **Add Input**. Enter K as the input. Right-click on the input terminal of the K variable and select **Create » Control** from the shortcut menu.
- ☐ Right-click on the left border of the MathScript Node and select **Add Input**. Enter Rm as the input. Right-click on the input terminal of the Rm variable and select **Create » Control** from the shortcut menu.
- ☐ Right-click on the left border of the MathScript Node and select **Add Input**. Enter J as the input. Right-click on the input terminal of the J variable and select **Create » Control** from the shortcut menu.
- ☐ Right-click on the right border of the MathScript Node and select **Add Output**. Enter sys as the output. Right-click on the output terminal of the sys variable and select **Choose Data Type » Add-ons » TF Object**.
- ☐ Wire the **sys** output terminal of the MathScript Node to the **Model 2** input terminal of the CD Series.vi.
- ☐ Right-click on the right border of the MathScript Node and select **Add Output**. Enter plant as the output. Right-click on the output terminal of the plant variable and select **Choose Data Type » Add-ons » TF Object**.
- ☐ Place the CD Draw Transfer Function Equation.vi on the block diagram. From the functions palette, navigate to **Control Design & Simulation » Control Design » Model Construction » CD Draw Transfer Function Equation.vi**, and place it on the block diagram.
- ☐ Wire the **plant** output terminal of the MathScript Node to the **Transfer Function Model** input terminal of the CD Draw Transfer Function.vi.
- ☐ Right-click the **Equation** output terminal of the CD Draw Transfer Function Equation.vi, and select **Create » Indicator** from the shortcut menu.
- ☐ Double-click the label of the picture indicator, and rename it Plant Equation.
- ☐ Create a control button for the **Stop** function of the While Loop.
- ☐ The block diagram should look like this:



☐ Save the VI (<Ctrl+S>).

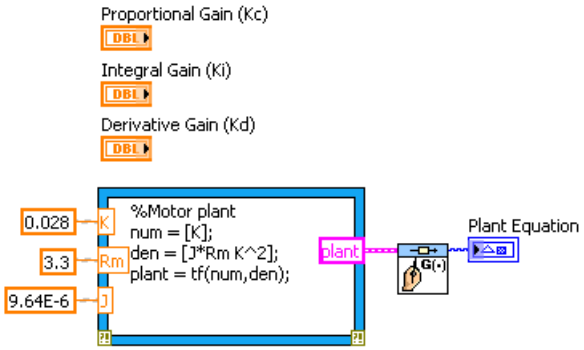
2. Run the VI
   - ☐ Switch to the front panel of the VI (<Ctrl+E>).
   - ☐ Set the front panel controls according to the following table:

K	0.028
Rm	3.3
J	9.64E-06
Proportional Gain (Kc)	1
Integral Gain (Ki)	0
Derivative Gain (Kd)	0.05

   - ☐ Set the front panel control values to default. Select **Edit » Make Current Values Default**.
   - ☐ Run the VI (<Ctrl+R>). Change the values of `Proportional Gain (Kc)`, `Integral Gain (Ki)`, and `Derivative Gain (Kd)`, and observe how the time and frequency response change.
   - ☐ Save the VI.

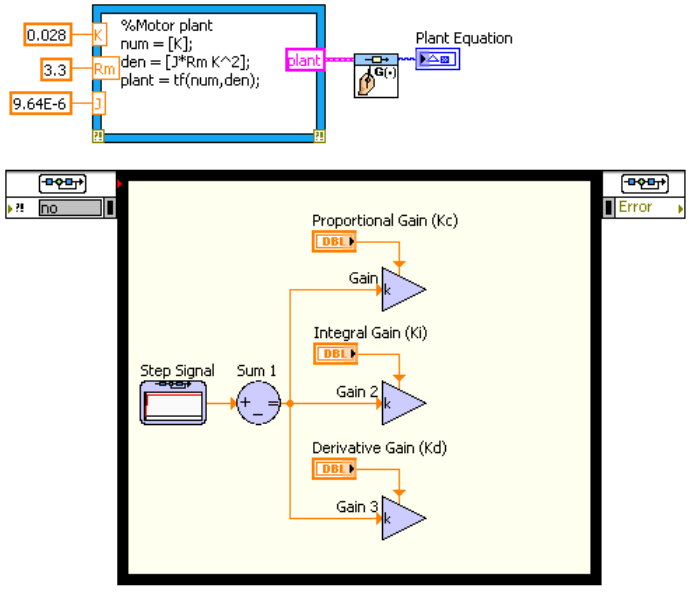
Exercise 3:

- ☐ Open `Exercise_2.vi`. Save the VI as `Exercise_3.vi`.
- ☐ Go to the block diagram. Remove the While Loop. Right-click the border of the loop and select **Remove While Loop** so that the code inside the loop is not deleted.
- ☐ Change the controls connected to the MathScript Node into constants.
  - o Right-click the K control and select **Change to Constant** from the shortcut menu. Set the value of the constant to 0.028.
  - o Right-click the `Rm` control and select **Change to Constant** from the shortcut menu. Set the value of the constant to 3.3.
  - o Right-click the `J` control and select **Change to Constant** from the shortcut menu. Set the value of the constant to 9.64E-6.
- ☐ Remove everything from the block diagram except for the MathScript Node, the constants wired to the MathScript Node, the PID controls (Proportional Gain (Kc), Integral Gain (Ki), and Derivative Gain (Kd)), and the `CD Draw Transfer Function Equation.vi`.
- ☐ Optional: Remove the `sys` output terminal from the MathScript Node, then remove lines 5-10 from the code inside the MathScript Node. (This portion of code creates the `sys` model, which consists of the `plant` model in series with an integrator. In this exercise, the integration will be performed in a Simulation Loop.)
- ☐ The block diagram should look like this:



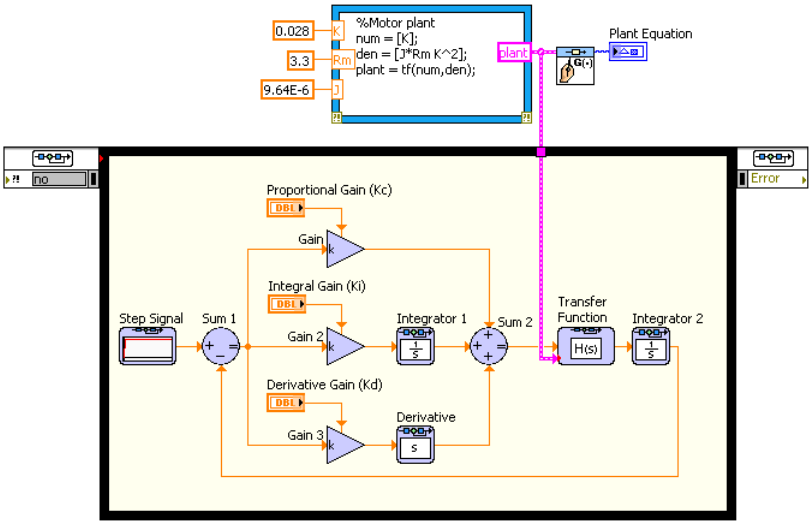
- ☐ Place a **Simulation Loop** on the block diagram. From the functions palette, navigate to **Control Design & Simulation » Simulation » Simulation Loop**. Draw a rectangle below the MathScript Node on the block diagram to define the loop.
- ☐ Place the `Proportional Gain (Kc)`, `Integral Gain (Ki)`, and `Derivative Gain (Kd)` controls inside the Simulation Loop.
- ☐ Place a **Step Signal** block inside the Simulation Loop. From the functions palette, navigate to **Control Design & Simulation » Simulation » Signal Generation » Step Signal**, and place it in the loop. Double-click the Step Signal block to open the configuration dialog. Set the value of **step time** to 0.01. Click **OK**.
- ☐ Place a **Summation** block inside the Simulation Loop. From the functions palette, navigate to **Control Design & Simulation » Simulation » Signal Arithmetic » Summation**, and place it in the loop. Label the Summation block "Sum 1".

- ☐ Place three **Gain** blocks inside the Simulation Loop. From the functions palette, navigate to **Control Design & Simulation » Simulation » Signal Arithmetic » Gain**, and place each block in the loop. Do the following for each Gain block:
  - Right-click the Gain block and select **Visible Items » Label** from the shortcut menu.
  - Right-click the Gain block and select **Configuration** from the shortcut menu. Set **Parameter Source** to "Terminal".
- ☐ Make the following wiring connections in the Simulation Loop:
  - Connect the **output** terminal of the Step Signal block to the **Operand1** terminal of the Sum 1 block.
  - Connect the **Result** terminal of the Sum 1 block to the **Input** terminal of each of the three Gain blocks.
  - Connect the `Proportional Gain (Kc)` control to the **gain** terminal of the Gain block.
  - Connect the `Integral Gain (Ki)` control to the **gain** terminal of the Gain 2 block.
  - Connect the `Derivative Gain (Kc)` control to the **gain** terminal of the Gain 3 block.
- ☐ The block diagram should look like this:



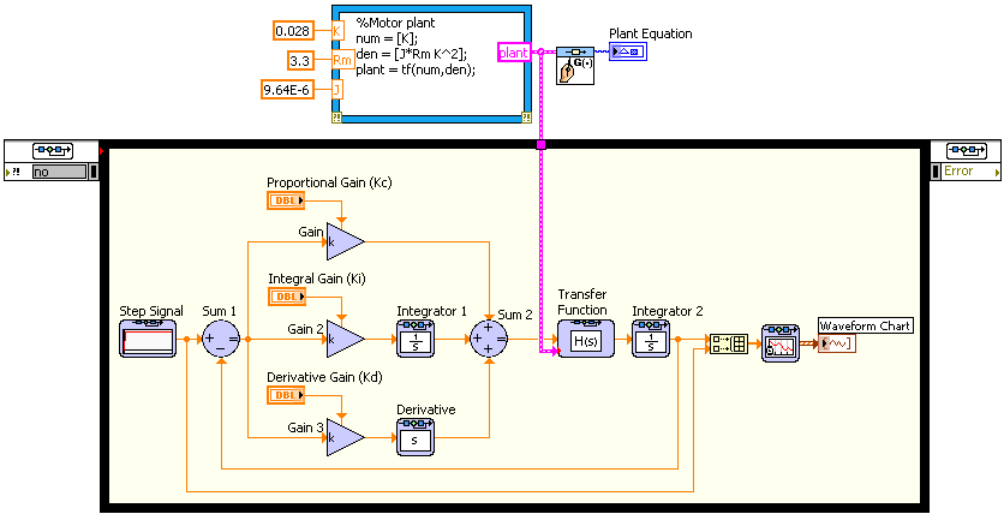
- ☐ Place two **Integrator** blocks inside the Simulation Loop. From the functions palette, navigate to **Control Design & Simulation » Simulation » Continuous Linear Systems » Integrator**, and place each block in the loop. Label the two blocks "Integrator 1" and "Integrator 2"
- ☐ Place a **Derivative** block inside the Simulation Loop. From the functions palette, navigate to **Control Design & Simulation » Simulation » Continuous Linear Systems » Derivative**, and place each block in the loop.

- ☐ Place another **Summation** block inside the Simulation Loop. Label the Summation block "Sum2". Double-click the Sum2 block to open the configuration dialog. Click the icons on the diagram so that the block has three "Add" input terminals. Click **OK**.
- ☐ Place a **Transfer Function** block inside the Simulation Loop. From the functions palette, navigate to **Control Design & Simulation » Simulation » Continuous Linear Systems » Transfer Function**, and place it in the loop. Double-click the Transfer Function block to open the configuration dialog. Set the **Parameter source** option to "Terminal". Click **OK**.
- ☐ Make the following wiring connections in the Simulation Loop:
    - o Connect the **output** terminal of the Gain 2 block to the **input** terminal of the Integrator 1 block.
    - o Connect the **output** terminal of the Gain 3 block to the **input** terminal of the Derivative block.
    - o Connect the **output** terminals of the Gain, Integrator 1, and Derivative blocks to the **Operand1**, **Operand2**, and **Operand3** terminals of the Sum 2 block.
    - o Connect the **Result** terminal of the Sum 2 block to the **input** terminal of the Transfer Function block.
    - o Connect the **plant** output of the MathScript Node to the **Transfer Function** terminal of the Transfer Function block.
    - o Connect the **output y(k)** terminal of the Transfer Function block to the **input** terminal of the Integrator 2 block.
    - o Connect the **output** terminal of the Integrator 2 block to the **Operand2** terminal of the Sum 1 block. This will close the feedback loop.
- ☐ The block diagram should look like this:



- ☐ Place a **Build Array** function inside the Simulation Loop. From the functions palette, navigate to **Programming » Array » Build Array**, and place it in the loop. Resize the Build Array node so that it has two inputs.

- ☐ Place a **SimTime Waveform** block inside the Simulation Loop. From the functions palette, navigate to **Control Design & Simulation » Simulation » Graph Utilities » SimTime Waveform**, and place it in the loop.
- ☐ Make the following wiring connections in the Simulation Loop:
  - Connect the **output** terminal of the Integrator 2 block to the first input terminal of the Build Array function.
  - Connect the **output** terminal of the Step Signal block to the second input terminal of the Build Array function.
  - Connect the **appended array** terminal of the Build Array function to the **Value** terminal of the SimTime Waveform block.
- ☐ Configure the simulation parameters of the Simulation Loop:
  - Right-click the border of the Simulation Loop, and select **Configure Simulation Parameters**.
  - Set the value of **Final Time (s)** to 1.
  - Set the value of **ODE Solver** to "Runge-Kutta 4".
  - Set the value of **Step Size (s)** to 0.001. This will set the simulation step size to one millisecond.
  - Click the **Timing Parameters** tab, and check the box labeled **"Synchronize Loop to Timing Source"**.
  - Set the value of **Period** to 1. This will set the loop to run once per millisecond (actual time).
  - Click **OK**.
- ☐ The block diagram should look like this:



- ☐ Optional: Configure the display options for the Waveform Chart.
  - Switch to the front panel of the VI (<Ctrl+E>).
  - Right-click the Waveform Chart, and select **Properties**.
  - In the **Display Format** tab, select "Floating Point" for the value of **Type**. Click **OK**.

- Double-click the rightmost numerical value on the x-axis of the Waveform Chart, and set the value to 1.

☐ Run the VI. Change the value of the PID Gains, and then run the VI again.

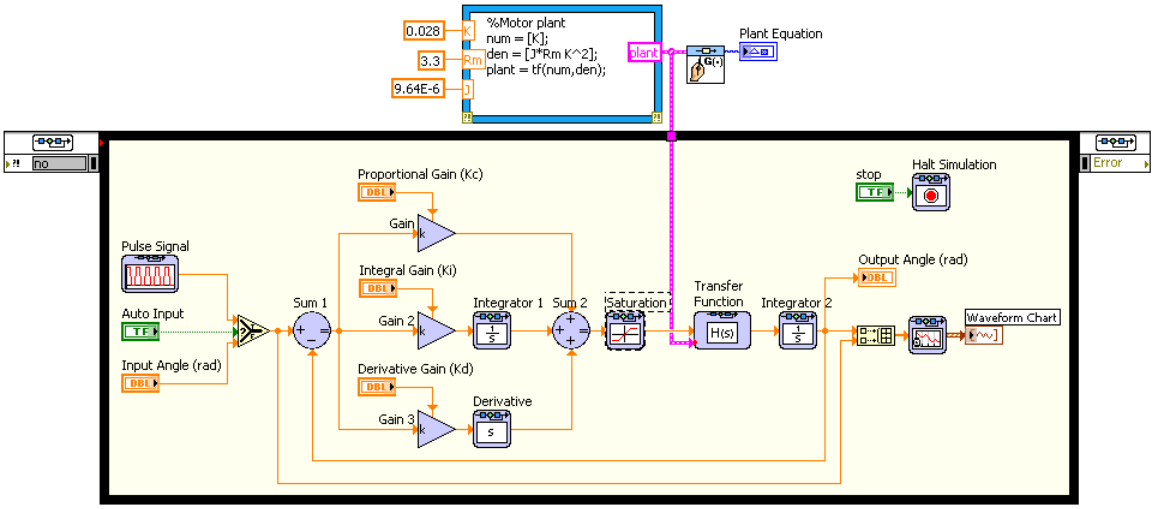
Exercise 2:

1. Build the VI block diagram
    - ☐ Open `Exercise_3.vi`. Save the VI as `Exercise_4.vi`.
    - ☐ Place a **Boolean** button on the front panel. From the controls palette, navigate to **Modern » Boolean » Push Button**, and place it on the front panel. Label the button "Auto Input".
    - ☐ Place a **Dial** on the front panel. From the controls palette, navigate to **Modern » Numeric » Dial**, and place it on the front panel. Label the dial "Input Angle (rad)".
    - ☐ Set the minimum and maximum values of the scale of the "Input Angle (rad)" dial to -3.14 and 3.14, respectively.
    - ☐ Make a copy of the "Input Angle (rad)" dial (<Ctrl+Drag>). Label the new dial "Output Angle (rad)".
    - ☐ Change the "Output Angle (rad)" dial from a control to an indicator. Right-click the dial and select **Change to Indicator** from the shortcut menu.
    - ☐ Place a **Stop Button** on the front panel. From the controls palette, navigate to **Modern » Boolean » Stop Button**, and place it on the front panel. This will allow you to terminate the simulation at will.
    - ☐ Take the following steps to customize the Waveform Chart display options:
        - o Right-click the chart and select **Y Scale » Autoscale Y** from the shortcut menu. Verify that the Autoscale Y option is no longer checked.
        - o Set the minimum and maximum values of the Y Axis to -4 and 4, respectively.
        - o Set the maximum value of the X Axis to 2.
    - ☐ Switch to the block diagram of the VI (<Ctrl+E>).
    - ☐ Right-click the border of the Simulation Loop, and select **Configure Simulation Parameters** from the shortcut menu. Set the value of **Final time (s)** to "inf". This will set the Simulation Loop to run until it is manually terminated. Click **OK**.
    - ☐ Make sure that the "Auto Input", "Input Angle (rad)", "Output Angle (rad)", and "stop" control terminals are all inside the Simulation Loop.
    - ☐ Right-click the **Step Signal** block, and select **Replace » Programming » Comparison » Select** from the shortcut menu. This replaces the Step Signal block with a **Select** function.
    - ☐ Place a **Pulse Signal** block inside the Simulation Loop. From the functions palette, navigate to **Control Design & Simulation » Simulation » Signal Generation » Pulse Signal**, and place it in the loop.
    - ☐ Double-click the Pulse Signal block to open the configuration dialog. Set the value of **amplitude** to 3.14. Click **OK**.
    - ☐ Place a **Halt Simulation** block inside the Simulation Loop. From the functions palette, navigate to **Control Design & Simulation » Simulation » Utilities » Halt Simulation**, and place it in the loop.
    - ☐ Make the following wiring connections in the Simulation Loop:

- Wire the **output** terminal of the Pulse Signal block to the **t** (top-left) terminal of the Select function.
- Wire the “Input Angle (rad)” control terminal to the **f** (bottom-left) terminal of the Select function.
- Wire the “Auto Input” control terminal to the **s** (center-left) terminal of the Select function.
- Wire the **output** terminal of the Integrator 2 block to the ”Output Angle (rad)” indicator terminal.
- Wire the “stop” control terminal to the **Halt?** terminal of the Halt Simulation block.

☐ Insert a **Saturation** block at the PID output. Right-click the wire connecting the **Result** terminal of the Sum 2 block to the **input u(k)** terminal of the Transfer Function block. Select **Insert » All Palettes » Control Design & Simulation » Simulation » Nonlinear » Saturation**. Double-click the Saturation block to open the configuration dialog. Set the values of **lower limit** and **upper limit** to -24 and 24, respectively (These values represent the maximum voltage inputs of the DC motor). Click **OK**.

☐ The block diagram should look like this:



☐ Save the VI (<Ctrl+S>).

2. Run the VI
   - ☐ Switch to the front panel of the VI (<Ctrl+E>).
   - ☐ Run the VI (<Ctrl+R>). Click the “Auto Input” button, and observe the output signal follow a pulse signal.
   - ☐ Turn off the “Auto Input” button, and change the value of “Input Angle (rad)”. The output signal should respond.
   - ☐ Change the values of the PID gains, and observe how the output signal responds to the input with different gain values.

- ☐ Press the “Stop” button to end the simulation.